# 0. Synchronized Multimedia Integration Language (SMIL) Document Object Model

Previous version:
http://www.w3.org/AudioVideo/Group/DOM/smil-dom-990721.html (W3C members only)
Editors:
Patrick Schmitz (Microsoft),
Jin Yu (Compaq),
Nabil Layaïda (INRIA)

## Abstract

This is a working draft of a Document Object Model (DOM) specification for synchronized multimedia functionality. It is part of work in the Synchronized Multimedia Working Group (SYMM) towards a next version of the SMIL language and SMIL modules. Related documents describe the specific application of this SMIL DOM for SMIL documents and for HTML and XML documents that integrate SMIL functionality. The SMIL DOM builds upon the Core DOM functionality, adding support for timing and synchronization, media integration and other extensions to support synchronized multimedia documents.

## Table of Contents

# 1.0 Introduction

The first W3C Working Group on Synchronized Multimedia (SYMM) developed SMIL - Synchronized Multimedia Integration Language. This XML-based language is used to express synchronization relationships among media elements. SMIL 1.0 documents describe multimedia presentations that can be played in SMIL-conformant viewers.

SMIL 1.0 did not define a Document Object Model. Because SMIL is XML based, the basic functionality defined by the Core DOM is available. However, just as HTML and CSS have defined DOM interfaces to make it easier to manipulate these document types, there is a need to define a specific DOM interface for SMIL functionality. The current SYMM charter includes a deliverable for a SMIL-specific DOM to address this need, and this document specifies the SMIL DOM interfaces.

Broadly defined, the SMIL DOM is an Application Programming Interface (API) for SMIL documents and XML/HTML documents that integrate SMIL functionality. It defines the logical structure of documents and the way a document is accessed and manipulated. This is described more completely in "What is the Document Object Model".

The SMIL DOM will be based upon the DOM Level 1 Core functionality. This describes a set of objects and interfaces for accessing and manipulating document objects. The SMIL DOM will also include the additional event interfaces described in the DOM Level 2 Events specification. The SMIL DOM extends these interfaces to describe elements, attributes, methods and events specific to SMIL functionality. Note that the SMIL DOM does not include support for DOM Level 2 Namespaces, Stylesheets, CSS, Filters and Iterators, and Model Range specifications.

The SYMM Working Group is also working towards a modularization of SMIL functionality, to better support integration with HTML and XML

applications. Accordingly, the SMIL DOM is defined in terms of the SMIL modules.

## 2.0 Requirements

The design and specification of the SMIL DOM must meet the following set of requirements.

General requirements:

- Inherit DOM Level 1 core functionality - the SMIL DOM will be based upon the generic Core DOM foundation.
- Support constraints on DOM core functionality (e.g. mutation), especially at runtime.
- Support both SMIL documents as well as hybrid documents that integrate XML/HTML and SMIL functionality.
- Support and reflect the modularization of SMIL functionality. It must be possible to design hybrid documents combining XML/HTML and SMIL functionality, that can in turn support a hybrid or combined DOM.

SMIL specific requirements

- Support SMIL specific elements. It must be possible to access and manipulate all SMIL elements with a SMIL document or a hybrid document that integrates XML and SMIL modules.
- Support SMIL specific attributes and methods. It must be possible to access and manipulate the SMIL attributes and methods on SMIL elements as well as XML/HTML elements in a hybrid documents.
- Support SMIL specific events, including:
  - Specification of statically defined SMIL events
  - Support for dynamically defined events
  - The ability to create and raise all the above events
- Support SMIL semantics. This includes various constraints on document structure, attribute values and method invocation.
- Define basic control interface for media player/renderers. Do not define a plug-in API, but rather just the timing and sync control interface.

It is not yet clear what all the requirements on the SMIL DOM will be related to the modularization of SMIL functionality. While the HTML Working Group is also working on modularization of XHTML, a modularized HTML DOM is yet to be defined. In addition, there is no general mechanism yet defined for combining DOM modules for a particular profile.

## 3.0 Core DOM: the SMIL DOM Foundation

The SMIL DOM has as its foundation the Core DOM. The SMIL DOM includes the support defined in the DOM Level 1 Core API, and the DOM Level 2 Events API.

## 3.1 DOM Level 1 Core

The DOM Level 1 Core API describes the general functionality needed to manipulate hierarchical document structures, elements and attributes. The SMIL DOM describes functionality that is associated with or depends upon SMIL elements and attributes. Where practical, we would like to simply inherit functionality that is already defined in the DOM Level 1 Core. Nevertheless, we want to present an API that is easy to use, and familiar to script authors that work with the HTML and CSS DOM definitions.

Following the pattern of the HTML DOM, the SMIL DOM follows a naming convention for properties, methods, events, collections and data types. All names are defined as one or more English words concatenated together to form a single string. The property or method name starts with the initial keyword in lowercase, and each subsequent word starts with a capital letter. For example, a method that converts a time on an element local timeline to global document time might be called "localToGlobalTime".

**Properties and methods**

In the ECMAScript binding, properties are exposed as properties of a given object. In Java, properties are exposed with get and set methods.

Most of the properties are directly associated with attributes defined in the SMIL syntax. By the same token, most *(or all?)* of the attributes defined in the SMIL syntax are reflected as properties in the SMIL DOM. There are also additional properties in the DOM that present aspects of SMIL semantics (such as the current position on a timeline).

The SMIL DOM methods support functionality that is directly associated with SMIL functionality (such as control of an element timeline).

*Note that the naming follows the DOM standard for XML, HTML and CSS DOM. This matches the HTML attribute naming scheme, but is on conflict with the SMIL 1.0 (and CSS) attribute naming conventions (all-lower with dashes between words). Given that the DOM Level 2 CSS API follows the primary DOM naming conventions, I think we should as well. Although this presents a naming conflict with the SMIL attributes (unless we reconsider attribute naming in the next version of SMIL), it presents a consistent DOM API.*

**Constraints on Core interfaces**

In some instances, the SMIL DOM defines constraints on the Level 1 Core interfaces. These are introduced to simplify the SMIL - associated runtime engines. The constraints include:

- Read-only properties, precluding arbitrary manipulation of the SMIL element properties at runtime.
- Disallowed structural changes, precluding certain changes to the structure of the document (and the associated time graph) at runtime.

These constraints are defined in detail below.

*This section will need to be reworked once we have a better handle on the approach we take (w.r.t. modality, etc.) and the details of the interfaces.*

*We probably also want to include notes on the recent discussion of a presentation or runtime object model as distinct from the DOM.*

## 3.2 DOM Level 2 Event Model

One of the goals of DOM Level 2 Event Model is the design of a generic event system which allows registration of event handlers, describes event flow through a tree structure, and provides basic contextual information for each event. The SMIL event model includes the definition of a standard set of events for synchronization control and presentation change notifications, a means of defining new events dynamically, and the defined contextual information for these events.

### 3.2.1 SMIL and DOM events

The DOM Level 2 Events specification currently defines a base Event interface and three broad event classifications:

- UI events
- UI Logical events
- Mutation events

In HTML documents, elements generally behave in a passive (or sometimes reactive) manner, with most events being user-driven (mouse and keyboard events). In SMIL, all timed elements behave in a more active manner, with many events being content-driven. Events are generated for key points or state on the element timeline (at the beginning, at the end and when the element repeats). Media elements generate additional events associated with the synchronization management of the media itself.

The SMIL DOM makes use of the general UI and mutation events, and also defines new event types, including:

- Object Temporal Events
- Logical Temporal Events
- Synchronization Events
- Media-delivery Events

Some runtime platforms will also define new UI events, e.g. associated with a control unit for web-enhanced television (e.g. channel change and simple focus navigation events). In addition, media players within a runtime may also define specific events related to the media player (e.g. low memory).

The SMIL events are grouped into four classifications:

Static SMIL events
: This is a group of events that are required for SMIL functionality. Some of the events have more general utility, while others are specific to SMIL modules and associated documents (SMIL documents as well as HTML and XML documents that integrate SMIL modules).

Platform and environment specific events
: These events are not defined in the specification, but may be created and raised by the runtime environment, and may be referenced by the SMIL syntax.

Author-defined events
: This is a very important class of events that are not specifically defined in the DOM, but that must be supported for some common use-case scenarios. A common example is that of broadcast or streaming media with embedded triggers. Currently, a media player exposes these triggers by calling script on the page. To support purely declarative content, and to support a cleaner model for script integration, we allow elements to raise events associated with these stream triggers. The events are identified by names defined by the author (e.g. "onBillWaves" or "onScene2"). Declarative syntax can bind to these events, so that some content can begin (or simply appear) when the event is raised. This is very important for things like Enhanced Television profiles, Enhanced DVD profiles, etc.
This functionality is built upon the DOM Level 2 Events specification.

Property mutation events
: These are mutation events as defined in the DOM Level 2 Events specification. These events are raised when a particular property is changed (either externally via the API, or via internal mechanisms).

### 3.2.2 Event propagation support

In addition to defining the basic event types, the DOM Level 2 Events specification describes event flow and mechanisms to manipulate the event flow, including:

- Event Capturing
- Event Bubbling
- Event Cancellation

The SMIL DOM defines the behavior of Event capture, bubbling and cancellation in the context of SMIL and SMIL-integrated Documents.

In the HTML DOM, events originate from within the DOM implementation, in response to user interaction (e.g. mouse actions), to document changes or to some runtime state (e.g. document parsing). The DOM provides methods to register interest in an event, and to control event capture and bubbling. In particular, events can be handled locally at the target node or centrally at a particular node. This support is included in the SMIL DOM. Thus, for example, synchronization or media events can be handled locally on an element, or re-routed (via the bubbling mechanisms) to a parent element or even the document root. Event registrants can handle events locally or centrally.

Note: It is currently not resolved precisely how event flow (dispatch, bubbling, etc.) will be defined for SMIL timing events. Especially when the timing containment graph is orthogonal to the content structure (e.g. in XML/SMIL integrated documents), it may make more sense to define timing event flow relative to the timing containment graph, rather than the content containment graph. This may also cause problems, as different event types will behave in very different ways within the same document.

Note: It is currently not resolved precisely how certain user interface events (e.g. onmouseover, onmouseout) will be defined and will behave for SMIL documents. It may make more sense to define these events relative to the regions and layout model, rather than the timing graph.

# 4.0 Constraints imposed upon DOM

We have found that the DOM has utility in a number of scenarios, and that these scenarios have differing requirements and constraints. In particular, we find that editing application scenarios require specific support that the browser or runtime environment typically does not require. We have identified the following requirements that are directly associated with support for editing application scenarios as distinct from runtime or playback scenarios:

## 4.1 Document modality

Due to the time-varying behavior of SMIL and SMIL-integrated document types, we need to be able to impose different constraints upon the model depending upon whether the environment is editing or browsing/playing back. As such, we need to introduce the notion of modality to the DOM (and perhaps more generally to XML documents). We need a means of defining modes, of associating a mode with a document, and of querying the current document mode.

We are still considering the details, but it has been proposed to specify an active mode that is most commonly associated with browsers, and a non-active or editing mode that would be associated with an editing tool when the author is manipulating the document structure.

### 4.2 Node locking

Associated with the requirement for modality is a need to represent a lock or read-only qualification on various elements and attributes, dependent upon the current document mode.

For an example that illustrates this need within the SMIL DOM: To simplify runtime engines, we want to disallow certain changes to the timing structure in an active document mode (e.g. to preclude certain structural changes or to make some properties read-only). However when editing the document, we do not want to impose these restrictions. It is a natural requirement of editing that the document structure and properties be mutable. We would like to represent this explicitly in the DOM specification.

There is currently some precedent for this in HTML browsers. E.g. within Microsoft Internet Explorer, some element structures (such as tables) cannot be manipulated while they are being parsed. Also, many script authors implicitly define a "loading" modality by associating script with the document.onLoad event. While this mechanism serves authors well, it nevertheless underscores the need for a generalized model for document modality.

### 4.3 Grouped, atomic changes

A related requirement to modality support is the need for a simplified transaction model for the DOM. This would allow us to make a set of logically grouped manipulations to the DOM, deferring all mutation events and related notification until the atomic group is completed. We specifically do not foresee the need for a DBMS-style transaction model that includes rollback and advanced transaction functionality. We are prepared to specify a simplified model for the atomic changes. For example, if any error occurs at a step in an atomic change group, the atomicity can be broken at that point.

As an example of our related requirements, we will require support to optimize the propagation of changes to the time-graph modeled by the DOM. A typical operation when editing a timeline shortens one element of a timeline by trimming material from the beginning of the element. The associated changes to the DOM require two steps:

- Change the begin time of the element to be later in time
- Change the duration of the element to preserve the end time

Typically, a timing engine will maintain a cache of the global begin and end times for the elements in the timeline. These caches are updated when a time that they depend on changes. In the above scenario, if the timeline represents a long sequence of elements, the first change will propagate to the whole chain of time-dependents and recalculate the cache times for all these elements. The second change will then propagate, recalculating the cache times again, and restoring them to the previous value. If the two operations could be grouped as an atomic change, deferring the change notice, the cache mechanism will see no effective change to the end time of the original element, and so no cache update will be required. This can have a significant impact on the performance of an application.

When manipulating the DOM for a timed multimedia presentation, the efficiency and robustness of the model will be greatly enhanced if there is a means of grouping related changes and the resulting event propagation into an atomic change.

# 5.0 SMIL specific extensions

*In all the interfaces below, the details need discussion and review. Do not assume that the defined types, return values or exceptions described are final.*

*The IDL interfaces will be moved to specific module documents once they are ready.*

## 5.1 Document Interface

Cover document timing, document locking?, linking modality and any other document level issues. Are there issues with nested SMIL files?

*Is it worth talking about different document scenarios, corresponding to differing profiles? E.g. Standalone SMIL, HTML integration, etc.*

## 5.2 SMIL Interfaces

*A separate document should describe the integrated DOM associated with SMIL documents, and documents for other document profiles (like*

*HTML and SMIL integrations).*

The SMILElement interface is the base for all SMIL element types. It follows the model of the HTMLElement in the HTML DOM, extending the base Element class to denote SMIL-specific elements.

Note that the SMILElement interface overlaps with the HTMLElement interface. In practice, an integrated document profile that include HTML and SMIL modules will effectively implement both interfaces (see also the DOM documentation discussion of Inheritance vs Flattened Views of the API).

**Interface SMILElement**

Base interface for all SMIL elements.

```
interface SMILElement : Element {
        attribute  DOMString            id;
        // etc. This needs attention
}
```

### 5.2.1 Structure Elements Interface

This module includes the SMIL, HEAD and BODY elements. These elements are all represented by the core SMIL element interface.

### 5.2.2 Meta Elements Interface

This module includes the META element.

**Interface SMILMetaElement (<meta>)**

```
interface SMILMetaElement : SMILElement {
        attribute  DOMString            content;
        attribute  DOMString            name;
        attribute  DOMString            skipContent;
        // Types may be wrong - review

}
```

### 5.2.3 Layout Interfaces

This module includes the LAYOUT, ROOT_LAYOUT and REGION elements, and associated attributes.

**Interface SMILLayoutElement (<layout>)**

Declares layout type for the document. See the LAYOUT element definition in SMIL 1.0

```
interface SMILLayoutElement : SMILElement {
        attribute  DOMString            type;
```

```
        // Types may be wrong - review
}
```

**Interface SMILRootLayoutElement (<root-layout>)**

Declares layout properties for the root element. See the ROOT-LAYOUT element definition in SMIL 1.0

```
interface SMILRootLayoutElement : SMILElement {
        attribute  DOMString            backgroundColor;
        attribute  long                 height;
        attribute  DOMString            skipContent;
        attribute  DOMString            title;
        attribute  long                 width;
        // Types may be wrong - review
}
```

**Interface SMILRegionElement (<region>)**

Controls the position, size and scaling of media object elements. See the REGION element definition in SMIL 1.0

```
interface SMILRegionElement : SMILElement {
        attribute  DOMString            backgroundColor;
        attribute  DOMString            fit;
        attribute  long                 height;
        attribute  DOMString            skipContent;
        attribute  DOMString            title;
        attribute  DOMString            top;
        attribute  long                 width;
        attribute  long                 zIndex;
        // Types may be wrong - review
}
```

The layout module also includes the region attribute, used in SMIL layout to associate layout with content elements. This is represented as an individual interface, that is supported by content elements in SMIL documents (i.e. in profiles that use SMIL layout).

**Interface SMILRegionInterface**

Declares rendering surface for an element. See the region attribute definition in SMIL 1.0

```
interface SMILRegionInterface {
        attribute  SMILRegionElement    region;
}
```

### 5.2.4 Timing Interfaces

This module includes the PAR and SEQ elements, and associated attributes.

*This will be fleshed out as we work on the timing module. For now, we*

*will define a time leaf interface as a placeholder for media elements. This is just an indication of one possibility - this is subject to discussion and review.*

**Interface SMILTimeInterface**

Declares timing information for timed elements.

```
interface SMILTimeInterface {
        attribute  InstantType          begin;
        attribute  InstantType          end;
        attribute  DurationType         dur;
        attribute  DOMString            repeat;
        // etc. Types may be wrong - review

            // Presentation methods
        void                            beginElement();
        void                            endElement();
        void                            pauseElement();
        void                            resumeElement();
        void                            seekElement(in InstantType seekTo);
}
```

**Attributes**

begin
: The date value of the begin instant of this node, relative to the parent timeline.

end
: The date value of the end instant of this node, relative to the parent timeline.

duration
: The duration value of this node.

etc.

**Presentation Methods**

beginElement
: Causes this element to begin the local timeline (subject to sync constraints).

**Parameters**
None

**Return Value**
None

**Exceptions**
None

endElement
: Causes this element to end the local timeline (subject to sync constraints).

**Parameters**
None

**Return Value**

None

**Exceptions**

None

pauseElement

Causes this element to pause the local timeline (subject to sync constraints).

**Parameters**

None

**Return Value**

None

**Exceptions**

None

resumeElement

Causes this element to resume a paused local timeline. If the timeline was not paused, this is a no-op.

**Parameters**

None

**Return Value**

None

**Exceptions**

None

seekElement

Seeks this element to the specified point on the local timeline (subject to sync constraints). If this is a timeline, this must seek the entire timeline (i.e. propagate to all timeChildren).

**Parameters**

seekTo:

The desired position on the local timeline.

**Return Value**

None

**Exceptions**

None

**Events**

onBegin

This event is raised when the element local timeline begins to play. It will be raised each time the element begins (but not on repeats - see the onRepeat Event). It may be raised both in the course of normal (i.e. scheduled) timeline play, as well as in the case that the element was begun with the beginElement() method. Note that if an element is not yet ready to play (e.g. if media is not ready), the onBegin event should not be raised until the element timeline actually begins to play and local time begins to advance.

As a composite timeline begins to play, each element will raise an onBegin event as it in turn begins to play. A parent element

will raise an onBegin event before any child elements do.

onEnd

This event is raised when the element local timeline ends play. It will be raised when the element ends (NOT on each repeat - see the onRepeat event). It may be raised both in the course of normal (i.e. scheduled) timeline play, as well as in the case that the element was ended with the endElement() method. As a composite timeline ends play, each element will raise an onEnd event as it in turn ends play.

onRepeat

This event is raised when the element local timeline repeats. It will be raised each time the element repeats, after the first iteration.

*This event should support an integer attribute to indicate the current repeat iteration.*

onPause

This event is raised when the element local timeline is paused. This is only raised when the element pauseElement() method is invoked.

*When pausing a timeline, I do not think that all descendents should also raise onPause events. However, it may be useful to have media descendents raise an onPause event. This needs attention.*
*I think we should consider supporting a "reason" attribute on this event. This would allow authors to disambiguate a pause due to a method call, and a pause forced by the timing engine as part of handling an out-of-sync problem.*

onResume

This event is raised when the element local timeline resumes after being paused. This is only raised when the element resumeElement() method is invoked, and only if the element was actually paused. *When resuming a timeline, I do not think that all descendents should also raise onResume events. However, it may be useful to have media descendents raise an onResume event. This needs attention.*

onOutOfSync

This event is raised when an element timeline falls out of sync (either for internal or external reasons). The default action of the timing model is to attempt to reestablish the synchronization, however the means may be implementation dependent. Depending upon the synchronization rules, this event may propagate up the time graph for each timeline that is affected.

onSyncRestored

This event is raised when an element that has fallen out of sync has been restored to the proper sync relationship with the parent timeline. This will only be raise after an onOutOfSync event has been raised. Depending upon the synchronization rules, this event

may propagate down the time graph, effectively "unwinding" the original onOutOfSync stack.

**Interface SMILTimelineInterface**

This is a placeholder - subject to change. This represents generic timelines.

```
interface SMILTimelineInterface : SMILTimeInterface {
        attribute NodeList          timeChildren;

        // Presentation methods

        NodeList                    getActiveChildrenAt();
        NodeList                    getActiveChildrenAt(
                                        in instant InstantType instant);
}
```

### Attributes

timeChildren

A NodeList that contains all timed children of this node. If there are no timed children, this is a Nodelist containing no nodes.

### Presentation Methods

getActiveChildrenAt

Causes this element to begin the local timeline (subject to sync constraints).

**Parameters**

instant: The desired position on the local timeline.

**Return Value**

NodeList: List of timed child-elements active at instant.

**Exceptions**

None

**Interface SMILParElement (<par>)**

```
interface SMILParElement : SMILTimelineInterface, SMILElement {
        attribute DOMString         endsync;
}
```

**Interface SMILSeqElement (<seq>)**

```
interface SMILSeqElement : SMILTimelineInterface, SMILElement {

}
```

### 5.2.5 Media Element Interfaces

This module includes the media elements, and associated attributes. They are all currently represented by a single interface, as there are no specific attributes for individual media elements.

**Interface SMILMediaInterface**

Declares media content.

```
interface SMILMediaInterface : SMILTimeInterface {
        attribute  DOMString            abstract;
        attribute  DOMString            alt;
        attribute  DOMString            author;
        attribute  ClipTime             clipBegin;
        attribute  ClipTime             clipEnd;
        attribute  DOMString            copyright;
        attribute  DOMString            fill;
        attribute  DOMString            longdesc;
        attribute  DOMString            src;
        attribute  DOMString            title;
        attribute  DOMString            type;
        // Types may be wrong - review
}
```

**Interface SMILRefElement (<ref>)**

```
interface SMILRefElement : SMILMediaInterface, SMILElement {

}

// audio, video, ...
```

### 5.2.6 Transition Interfaces

This module will include interfaces associated with transition markup. This is yet to be defined.

### 5.2.7 Animation Interfaces

This module will include interfaces associated with animation behaviors and markup. This is yet to be defined.

### 5.2.8 Linking Interfaces

This module includes interfaces for hyperlinking elements.

**Interface SMILAElement (<a>)**

Declares a hyperlink anchor. See the A element definition in SMIL

1.0.

```
interface SMILAElement : SMILElement {
            attribute  DOMString            title;
            attribute  DOMString            href;
            attribute  DOMString            show;
        // needs attention from the linking folks
}
```

### 5.2.9 Content Control Interfaces

This module includes interfaces for content control markup.

**Interface SMILSwitchElement (<switch>)**

Defines a block of content control. See the SWITCH element definition in SMIL 1.0

```
interface SMILSwitchElement : SMILElement {
        attribute  DOMString            title;
        // and...?
}
```

**Interface SMILTestInterface**

Defines the test attributes interface. See the Test attributes definition in SMIL 1.0

```
interface SMILTestInterface {
        attribute  DOMString            systemBitrate;
        attribute  DOMString            systemCaptions;
        attribute  DOMString            systemLanguage;
        attribute  DOMString            systemOverdubOrCaption;
        attribute  DOMString            systemRequired;
        attribute  DOMString            systemScreenSize;
        attribute  DOMString            systemScreenDepth;
        // and...?
}
```

## 5.3 Media Player Interfaces

*This is NOT a plug-in interface, but rather a simple interface that describes some guaranteed methods that any application plug-in interface must support. This provides a means of standardizing extensions to the timing model, independent of the specific application.*

**5.3.1 Media Player Level 1 Interface**

**5.3.2 Media Player Level 2 Interface**

**5.3.3 Media Player Level 3 Interface**

# 6.0 References


**[DOM-Level-1]**
"Document Object Model (DOM) Level 1 Specification"

Available at http://www.w3.org/TR/REC-DOM-Level-1/.

**[DOM2Events]**
"Document Object Model Events", T. Pixley, C. Wilson

Available at http://www.w3.org/TR/WD-DOM-Level-2/events.html.

**[DOMReqts]**
"Document Object Model Requirements for Synchronized Multimedia", P. Schmitz.

Available at http://www.w3.org/AudioVideo/Group/DOM/DOM_reqts (W3C members only).

**[HTML]**
"HTML 4.0 Specification", D. Raggett, A. Le Hors, I. Jacobs, 24 April 1998.

Available at http://www.w3.org/TR/REC-html40.

**[ISO/IEC 10646]**
ISO (International Organization for Standardization). ISO/IEC 10646-1993 (E). Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane. [Geneva]: International Organization for Standardization, 1993 (plus amendments AM 1 through AM 7).

**[PICS]**
"PICS 1.1 Label Distribution -- Label Syntax and Communication Protocols", 31 October 1996, T. Krauskopf, J. Miller, P. Resnick, W. Trees

Available at http://www.w3.org/TR/REC-PICS-labels-961031

**[RFC2045]**
"Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed and N. Borenstein, November 1996.

Available at ftp://ftp.isi.edu/in-notes/rfc2045.txt. Note that this RFC obsoletes RFC1521, RFC1522, and RFC1590.

**[SMIL]**
"Synchronized Multimedia Integration Language (SMIL) 1.0 Specification W3C Recommendation 15-June-1998 ".

Available at: http://www.w3.org/TR/REC-smil.

**[SMIL-CSS2]**
"Displaying SMIL Basic Layout with a CSS2 Rendering Engine".

Available at: http://www.w3.org/TR/NOTE-CSS-smil.html.

**[WAI]**
"WAI Accessibility Guidelines: User Agent", W3C Working Draft 3-July-1998.

Available at http://www.w3.org/WAI/UA/WD-WAI-USERAGENT.html

**[XML]**
"Extensible Markup Language (XML) 1.0", T. Bray, J. Paoli, C.M. Sperberg-McQueen, editors, 10 February 1998.

Available at http://www.w3.org/TR/REC-xml

---

previous contents

US-PAT-NO: 6427230

DOCUMENT-IDENTIFIER: **US 6427230 B1**

TITLE: System and method for defining and managing reusable groups software constructs within an object management system

--------- KWIC --------

US Document Identifier - DID (1):
**US 6427230 B1**

Brief Summary Text - BSTX (34):
According to another aspect of the invention, external interfaces associated with a package of software constructs may be recorded using interface specifications that are stored in the package objects. Each of these interface specifications describes the external interface of the associated package. That is, an interface specification describes the **input and output** parameters associated with the package of software constructs in a manner that allows a user to readily understand the functionality provided by the package. This functionality can be readily determined without having to understand the complex interdependencies between the software constructs included in the package. Using the interface specifications, a user may search for software packages that perform desired functions, thus making it easier to reuse these packages to develop new applications. These interface specifications may also be used, for example, by component consumers and component producers to identify a **requested** software function.

Drawing Description Text - DRTX (5):
FIG. 3 is a block diagram of the generalized model for the Element Inventory **Schema** (EIS);

Drawing Description Text - DRTX (6):
FIG. 4 is a block diagram showing the relationship between instances of elements and relationships, and the various type definitions provided in the model stored in the Element Inventory **Schema**;

Detailed Description Text - DETX (4):
According to the preferred embodiment, each element within the Element Inventory 102 is associated with a respective pre-defined element type. Examples of element types are "Table", "TableColumn", or "Program". These element types, which are recorded in a model associated with Element Inventory

102, are stored in the Element Inventory Schema (EIS) 103. The element type associated with a particular element is representative of the functionality of the associated data, code or system component. EIS further stores relationship types, which define how one element type is related to another element type. This will be discussed further below.

Detailed Description Text - DETX (5):

The Element Inventory 102 is supported using functions provided by Inventory Administration 104. These support functions include the backup facilities used to make a copy of selected elements, and restore facilities used to restore an existing saved copy of an element to the Element Inventory. The administration functions further include export and import operations provided to exchange information between the Element Inventories of one or more remote Object Management Systems such as that shown as Remote Object Manage System 107. The export function provides a copy of an element to the remote system, whereas the import function receives a copy of an element from a remote system, and stores the copy within the Element Inventory 102. Export/import exchanges are accomplished using self-defining intermediate file structures of the type utilized by various export/import standards such as eXtended Markup Language (XML).

Detailed Description Text - DETX (8):

The Element Inventory 102 and Inventory Administration 104 are managed by the Asset Inventory Manager 106. The Asset Inventory Manager (AIM) is the software that provides an interface to the Element Inventory 102 and Element Inventory Schema 103. One of the functions of the AIM is to hide the underlying repository implementation by providing an Application Program Interface (API) tailored for elements. The AIM provides an interface that supports the operations required by both the Mission Specific Facilities, shown in Block 108, and the Common Facilities, shown in Block 110.

Detailed Description Text - DETX (17):

Object Management System 100 further includes Common Facilities 110. These functions aid the user in understanding the relationships between elements, and also aid in invoking the tools used to perform the transformation and renovation operations. Common Facilities 110 include the Affinity Analyzer 122, which is a tool that analyzes the relationships existing between various elements contained within the Element Inventory 102. For example, the Affinity Analyzer determines which elements are involved in the processing performed to accomplish a particular function. The Affinity Analyzer 122 further provides a graphic display representing the various elements and element relationships for those code and data components provided by the IT platform. The graphical displays, which are capable of illustrating complex element networks, are used for advanced impact analysis and element packaging purposes. For example, before a particular code module is modified, the relationships existing between the associated element modeling that code module and other elements may be used to determine which other code or data components need to be modified to maintain compatibility. These relationships may be graphically depicted using

the Affinity Analyzer 122. The Affinity Analyzer allows software analysts to interrogate and visually mine single or disparate sets of elements without having to understand the details of the elements or relationships. **Query** and exploration, hypothesis generation, and knowledge discovery routines eliminate the need to compose complex **queries** for investigating how various code and data components are structured or interrelate. In the preferred embodiment, the Affinity Analyzer is implemented using the Netmap tool commercially available from the Alta Corporation.

Detailed Description Text - DETX (22):

A wide variety of vendor tools are available to perform Element Discovery Functions 142. For example, the Fulcrum tool commercially available from the RMC Limited Corporation (U.K.) is capable of automatically analyzing Cobol code constructs and related data structures. This tool must be synchronized with the element types included within a model and stored within the Element Inventory **Schema** 103. This synchronization allows Fulcrum to create elements having recognized element types and relationship types, and that are consistent with other element types used within the system. Many other types of tools are available for analyzing code and data constructs of various other types of software languages. The type of Element Discovery Functions 142 which are available within a given Object Management System 100 will vary depending on the types of IT functions and technologies that are supported by that system.

Detailed Description Text - DETX (25):

From the default Element Explorer View, the user is able to select one of the other views, including the Properties View, Relationships View, or Affinity View. The Properties View enables the user to view the list of attributes associated with a selected element or element type, and which are stored within the Element Inventory 102. Attributes provide additional information about the code and data module associated with, and described by, the selected element, as will be described further below. The Relationships View is a graphic illustration of the relationships existing between a selected element and other elements, or between a selected element type and other element types. This view further allows a user to create new relationships for that element. This will be described further below. In comparison to the Relationships View, the Affinity View provides a more distant view of a chain of related elements. The Affinity View takes as **input** a user-provided starting element or element type and an ending element or element type, and displays the starting element, the ending element, and all elements and element relationships which provide the chain, or "slice" of interconnections leading from the starting element to the ending element. A similar view is provided between a specified starting element type and a specified ending element type. The relationships which are provided in these views represent relationships between the code and data modules that are stored elsewhere, and that are modeled by the elements.

Detailed Description Text - DETX (35):

Within the ERM, the model that defines the various element types is called the Element Inventory **Schema** (EIS) 103, as discussed above. This model is

installed in the Element Repository at system installation time, but may be varied throughout the life of the system. The model definition may be specific to a particular Object Management System. The Element type definitions within EIS 103 provide the templates used during element creation, and define the type of information contained in, the attributes associated with, and the relationships existing between, the elements.

Detailed Description Text - DETX (39):

The UREP DLL 231 interfaces with the Asset Inventory Manager Executable (AIM EXE) 232. The AIM EXE implements the Asset Inventory Manager 106 function of FIG. 1. As discussed above, one of the functions of the AIM EXE 232 is to provide an interface to the Element Repository 220 that hides the underlying repository implementation. For example, the **services** provided by the AIM EXE hide the functions provided by the UREP DLL 231. The AIM EXE further masks the user from any transaction management and database locking that is required to accomplish a given task. The AIM EXE does so by providing an Application Program Interface (API) that supports the operations required by the entities accessing the various elements stored within the Element Inventory 102.

Detailed Description Text - DETX (40):

The following **services** are provided by the AIM EXE. Various ones of these **services** are called by the Element Discovery Functions 142, and Element Viewers 144 to perform the tasks discussed above: Connect: This **service** connects the **session** to the Element Repository. This **service** further opens the repository, makes a repository log entry in the newly created object, and begins a UREP **session**. Disconnect: This **service** disconnects the **session** from the Element Repository. In the preferred embodiment, this is accomplished by ending the UREP **session** and closing the repository. This **service** is called with a parameter that indicates whether uncommitted changes should be discarded. If uncommitted changes exist which are not to be discarded, the **request for this service** is disregarded. Export Element Types: This **service** reads element types from the EIS 103 and writes them into a file in a predetermined format as shown by dashed line 240. In the preferred embodiment, this format is **XML**. This **service** is called by scripts which execute on the Script Server 218. Import Element Types: This **service** reads element types from a file and writes them into the EIS 103 in a predetermined format, which in the preferred embodiment is **XML** format, and is shown by dashed line 240. This **service** is called by scripts that execute on the Script Server 218. The element types are installed at initialization time, and may be updated as desired during the life of a system. **Get** Element Types: This **service** reads element types from the EIS 103 and returns them to the caller in an **output** parameter. In the preferred embodiment, the **output** format is **XML**. Put Element Types: This **service** reads element types from an **input** parameter and writes them to the EIS 103. In the preferred embodiment, the **input** format is **XML**. Export Elements: This **service** reads elements from the Element Inventory 102 and writes them into a file as is indicated by dashed line 240. This **service** is called by scripts executing on either the Client Server 216 or the Script Server 218. Import Elements: A **service** which reads elements from a file and writes them into the Element Inventory 102 as indicated by dashed line 240. This **service** includes options

for handling already-existing elements, including the Ignore, Overwrite, and Create New Version options. This service is called by scripts executing on either the Client Server 216 or the Script Server 218. Get Elements: A service that reads elements from the Element Inventory 102 and returns them to the caller in an output parameter. This service is called by various ones of the Interactive Tools 259. The element that is to be retrieved may be specified according to an element name, or may be specified using relationship data used to address a particular element within the Element Inventory. Another option for this service allows an element to be specified for retrieval according to a particular character string that the element stores. Get Element for Update: A service called by various ones of the Interactive Tools. This service sets an update lock on an element for a particular session, then reads the selected element from the Element Inventory 102 so that it is returned to the requester as an output parameter. The selected element may be specified by element name, or may be specified using relationship data used to address an element within the Element Inventory. Another option allows the selected element to be specified according to particular character string that the element stores. Create Elements: A service called by the Interactive Tools, and that provides elements as input parameters so that they can be written to the Element Inventory 102. Update Element: A service called by the Interactive Tools 259 for providing elements as input parameters so that they can be written to the Element Inventory 102. This service must be preceded by a call to "Get Element for Update" service. Delete Elements: A service called by the Interactive Tools 259 that deletes specified elements from the Element Inventory 102. Get BLOB: A service called by Interactive Tools 259 that reads a Binary Large Object (BLOB) attribute from an Element in the Element Inventory 102 and writes it into a file. The file can reside on a remote host, specified by a Universal Naming Convention (UNC) name. Get BLOB for Update: A service called by Interactive Tools which sets an update lock for this session on a BLOB Element in the Element Inventory 102, reads its BLOB attribute, and writes the BLOB attribute into a file. The file can be on a remote host, specified by UNC name. Update BLOB: This service, which is called by the Interactive Tools, reads a BLOB from a file, and writes the BLOB as an attribute of a BLOB Element in the Element Repository 102. The file can be on a remote host, specified by UNC name. This service call must be preceded by Get BLOB for Update. Save: A service which commits all uncommitted changes to the Element Inventory 102. When invoked from interactive session, this service also saves a description of the state of the Common Workbench 111, including the state for any executing Interactive Tools. Undo Last: A service that rolls back the last uncommitted change to the Element Inventory 102. This service may be called by either Interactive Tools 259 or by scripts. Undo All: This service rolls back all uncommitted changes to the Inventory, if any such uncommitted changes exist. This service may be called by either the Interactive Tools 259 or scripts.

Detailed Description Text - DETX (41):

An instance of the AIM EXE 232 is created for each session that is active on the AIM server. If multiple sessions are active, multiple instances of the AIM EXE will be active at once. This is indicated by the replicated process indicator 244.

Detailed Description Text - DETX (42):

Creation of the AIM EXE 232 is performed by the **Session** Controller 248 as is indicated by control flow indicator 250. Creation of the AIM EXE invokes the "Connect" **service** to establish a **session** with the Element Repository 220. A **session** is ended when the **Session** Controller 248 calls the AIM EXE "Disconnect" **service**.

Detailed Description Text - DETX (43):

In the preferred embodiment, the **Session** Controller 248 is an NT **service** that is started automatically at AIM Server boot-up or manually by the administrator. The **Session** Controller is responsible for generating begin and end **session requests** in response to user **requests** received from Application Main 271 running on Client Server 216 to log in and log off the Object Management System, respectively. These **requests** are represented by control flow indicator 272. Such **requests** may also be received by the **Session** Controller 248 from script execution on Script Engine 273, as shown by control flow indicator 274. The **Session** Controller is also responsible for receiving administrator **requests** to terminate orphaned **sessions** by destroying orphaned COM objects.

Detailed Description Text - DETX (44):

The browser-level client interface to AIM Server 214 is provided by the Internet Information Server (IIS) 280. In the preferred embodiment, IIS 280 responds to **requests** from Web Browser 281 by delivering login Active Server Pages (ASPs) 282 to the user for allowing login functions to be performed. The **requests** are represented by control flow indicator 283, and the IIS responses are represented by data flow indicator 284. IIS returns an HTML page which displays the names of the Object Management Systems if more than one system is available to the user, and further directs the user to the **URL** of the selected login form ASP. The **URLs** for the login form ASP is obtained from NT Registry 285, which stores system persistent data such as system names and ASP **URLs** that cannot, or should not, be held in the Element Repository 220.

Detailed Description Text - DETX (45):

A login form ASP returns an HTML page to the client. This ASP contains a form in which the user enters login information such as a user id and password. The user is then directed to the Login **Request** ASP, which sends the entered information to the **Session** Controller 248 for validation, as shown by control flow indicator 286. If a **session** is not successfully created for the user, an HTML page is returned to the client **requesting** that the information be reentered. Otherwise, a **session** is established.

Detailed Description Text - DETX (46):

Once a user is logged in, the client-side application files for the Object Management System are downloaded from Mass Storage 287 to the Client Server

216, and Application Main 271 on Client Server begins execution. Thereafter, further communication between Client 216 and AIM Server 214 is performed via the Session Controller 248 using Distributed Component Object Module (DCOM) protocol, as shown by control flow indicator 272.

Detailed Description Text - DETX (47):

The AIM Server 214 further includes Plan Hub 288, which acts as a central router for script execution **requests** and script execution status. The Plan Hub receives **requests** for script execution from the Client Server 216, and forwards these **requests** to a specified Script Controller 289, which is the process executing on Script Server 218. These **requests** are represented by control flow indicator 290. In the preferred embodiment, Plan Hub is an NT **service**, is started automatically when the AIM Server is booted, and may also be started manually by a system administrator.

Detailed Description Text - DETX (48):

AIM Server 214 also includes Logger 292, which is an NT **service** that receives script execution status from Script Controller 289 so that this status can be recorded.

Detailed Description Text - DETX (50):

Turning now to a description of the Client Server 216, a user establishes a **session** with the AIM Server by invoking the **URLs** of Login ASPs using Web Browser 281. The user receives HTML login pages and application files from IIS 280. Once a user is logged in and a **session** is established for the user, communication between client and server is via DCOM, although the browser **session** is maintained.

Detailed Description Text - DETX (51):

Application Main 271 is the main client-side process supporting the object management functions. As an executable, the process for Application Main is started at creation. Application Main provides for the start-up of the individual Interactive Tools 259 via the interface shown as Line 260. Application Main further has access to ones of the global **services** such as Save, Undo, and Logoff, which are provided by the AIM EXE 232.

Detailed Description Text - DETX (52):

Invocation of Interactive Tools 259 by Application Main starts each of the Interactive Tools at its logical starting point. These tools provide the Mission-Specific Facilities 108, the Common Facilities 110, and the Common Workbench capabilities 111 of FIG. 1. These tools call **services** provided by the AIM EXE to perform functions such as retrieving element type definitions from the EIS 103, retrieving element data from the Element Inventory 102, making **requests** for the creation of new elements according to the EIS model, or making **requests** to modify existing elements. These **requests** are shown in FIG. 2 as Control Flow Indicator 262 and Data Flow Indicator 264. A new thread is

started when one of the Interactive Tools 259 begins communicating with the AIM EXE 232. When a **request** from Interactive Tools 259 is processed successfully by AIM EXE, notification of changes to data in the ER 220 is returned by the AIM EXE 232 via **Session** Controller 248 to Application Main 271, which then forwards the notification to Interactive Tools 259 via the interface shown as Line 260.

Detailed Description Text - DETX (54):

The AIM EXE 232 further communicates with Script Server 218. Within Script Server 218, Script Controller 289 accepts **requests** from the Plan Hub 288 to execute a Plan, which is a scripted invocation of **services** provided by AIM EXE 232, and/or programmatic tool invocations. In response, the Script Controller reads a **requested** scripted plan, which is stored as an element in the Element Inventory 102, via the AIM EXE 232 as indicated by control flow 266. Thereafter, Script Controller 289 forks a process in which the plan is executed by Script Engine 273. The Script Controller further sends periodic notifications to the Plan Hub 288 to report on the script execution status. Upon termination of the script, the Script Controller writes an element that is a plan completion record to the Element Inventory 102 and notifies the Plan Hub.

Detailed Description Text - DETX (55):

Detailed Description of the Element Inventory **Schema** and Domain Mappings

Detailed Description Text - DETX (56):

FIG. 3 is a block diagram of the generalized model for the Element Inventory **Schema** (EIS). As discussed above, the preferred embodiment of the Object Management System 100 utilizes a model loaded within EIS 103 which includes the element type definitions as represented by Element Type 302. Each element type may represent a particular type of software construct or data structure for a code or data module existing within one of the host systems interconnected to Object Management System 100. Examples of element types include "table", "program", "database", "application", and "transaction management system". As many element types may be defined as is necessary to support a given mission of the data processing systems that are managed by Object Management System 100. Generally, a set of element types will be loaded when the Object Management System is initialized. The model is flexible, and may be updated during the life of the system using **service** calls to the AIM EXE 232 such as "Put Element Types" and "Import Element Types".

Detailed Description Text - DETX (65):

FIG. 4 is a block diagram showing the relationship between instances of elements and instances of relationships, and the various type definitions provided in the model stored in the Element Inventory **Schema**. When an element, shown as Element 402, is created or loaded within Element Inventory 102, it is assigned to one of the predefined Element Types 302, as indicated by Line 403. Element 402 may be said to be an instance of that particular Element Type. By

virtue of this association, Element 402 acquires the potential to be related to other defined element types according to each predefined Binary Relationship Type 304 that is defined for the Element Type 302, as is shown by Lines 404 and 406. Element 402 also may become associated with an Attribute 408 that is an instance of Attribute Type 314 defined for Element Type 302, as is represented by Line 410.

Detailed Description Text - DETX (69):

FIG. 5 further depicts that an instance of element type "BaseTable" 502 exists called "Employee" 506. This element is a construct that stores data that describes the actual software module with which it is associated, which in this example is a table. That is, this element represents an actual table of data which exists elsewhere on the system, for example, on Host A 228. This element is assigned the element type of "BaseTable", and is also given a name representing the function provided by the table, which in this case is "Employee". The element further stores an indication of all other elements to which it is related. This actual table may include columns containing information about employees of a business. This is represented by the fact that element "Employee" is related to elements of type "Column" 504 that include elements "EmpID" 508, "LastNm" 510, and the like. These column elements are related to element"Table" through the Binary Relationship Type represent by Line 511, and described by roleA 404 and roleB 406 of "hasCol" 512 and "ofTbl" 514. This represents the relationship between the actual table component, and the individual column components that comprise the table. (For the sake of brevity, instances of binary relationships will be discussed from this point forward in terms of either roleA 404 or roleB 406, but not both.) Element type "BaseTable" is shown having various other predefined relationships to other element types include element type "View" 515. Two instances of element type "View" 515 shown as "EmPhoneDir" 516 and "EmpByState" 518 are shown to exist, each having a relationship with element "Employee" 506 represented by Line 507. Other element types related to element type "BaseTable" include "Key" 520, "Index" 522, "File" 524, "**Schema**" 526, and "Trigger" 528, all of which have Element instances created and stored in the Element Inventory 102 which are related to element Employee 506, as is represented by Lines 530 and 532, 534, 536, 538, and 540, respectively. It may be noted that element type "Constraint" 542 is related to element type "BaseTable". However, for the particular element instance "Employee" of element type "BaseTable", no element instance of type "Constraint" 542 has been created. This demonstrates that the element and relationship type definitions within EIS 103 define potential, not mandatory, relationships which may or may not be established for a given element instance. It may further be noted that more than one type of relationship can be established between an element of a given element type and another element of a second element type. For example, element "Employee" 506 of element type "BaseTable" 502 is capable of establishing the two different types of relationships represented by Lines 530 and 532 with element "EmpID" 544 of element type "Key" 520.

Detailed Description Text - DETX (75):

The use of the model in searching the Element Inventory 102 is best shown by

example. Assume the model for element type "BaseTable" is stored in the Element Inventory Schema 103. A particular named element within the Element Inventory may be known or is otherwise located. The associated element type for the known element is used to access the element type definition within the model. The model indicates that the subject element type is associated with one or more Binary Relationship Types. For each of these Binary Relationships Types, the meta-data stored within the element is analyzed to determine if any instance of these Binary Relationship Types exists for the particular element instance. If an instance of any of these Binary Relationship Types exists for the element instance, the Binary Relationship instance may be traversed to locate a related element. This process can be made recursive such that each of the identified elements is made the target of the search, and all related elements associated with the target are further identified. Using this method, the element relationships can be traversed to find all elements that are either directly, or indirectly, related to the target element. All relationship traversal is performed using the model definition. This model-based method is particularly useful because it can accommodate the situation wherein new element type definitions are added to the Element Inventory Schema 103, or wherein element type definitions are modified. This modification of element definitions can be accomplished without modifying any of the element management tools.

Detailed Description Text - DETX (82):

FIG. 6 is an illustration of the user interface for Element Explorer View provided by Element Viewers 144. The left-hand display window 600 provides the user with a list of element types defined within the EIS 103. This list is obtained by making a service call such as "Get Element Types" to the AIM DLL. Block 602 shows the element type "Table" being selected by the user. The other element types shown in FIG. 5 such as Column 504 are also shown in the left-hand display window of FIG. 6. Within Object Management System 100, hundreds or even thousands of element types may be defined. Window 600 also shows the element type hierarchy, with each element type having the ability to include one or more element subtypes below it in the hierarchy, which are illustrated in an indented format. In FIG. 6, element type "Table" 602 is shown including the element subtypes "BaseTable" 502 and "View" 515, which appear indented below it in the display.

Detailed Description Text - DETX (89):

Element Packager 118 requires as **input** parameters one or more Asset Elements that provide the foundation for constructing the Element Package. Element Packager then provides a display of all elements having relationships to the initially-specified elements. This display is obtained by first making a "Get Element" service call to the AIM DLL with the names of the initially-specified elements, for example, element "Employee". This causes the information for these elements to be provided to Element Packager 118. The Element Packager retrieves the relationship data from these structures. In the preferred embodiment, the relationship data is stored in the UREP fully object-oriented Element Repository 220 as pointer data between elements, and is returned to the user described as relationships. The relationship data is retrieved from

Element Repository 220 and provided to a **requesting** tool such as Element Viewers 144 after a **service** call such as "**Get** Element" or "**Get** Element for Update" which returns a **requested** element structure, along with the relationship data stored therein.

Detailed Description Text - DETX (103):

After the user has obtained the complete set of related elements and element relationships for the initially-selected element(s), the process is repeated. That is, the user is able to select the "Continue Button" 706 so that the process is repeated for the elements remaining in Column 800. As a result, all elements related to the elements in Column 800 are retrieved from the Element Inventory 102 via calls to **services** in the AIM EXE in the manner discussed above.

Detailed Description Text - DETX (107):

FIG. 10 is a flowchart of the iterative process used to traverse element relationships to obtain a defined Element Package. From Start Block 1002, an initial set of one or more elements must be identified, as shown by Step 1004. This could be done using a variety of tools including Element Viewers 144 and Element Locator 124, as discussed above. Once this initial element set is defined, each of the elements in the set is read from the Element Inventory 102 using **service "Get** Element" described above. For each element, all element relationships are identified using the relationship data included in the element structure as shown in Step 1006. In the preferred embodiment, this is accomplished by reading, for each element identified in Step 1004, the element type definition from the EIS 103, then using this element type definition to match a relationship instance stored in an element instance with a relationship type. Next, each of the identified relationships is used to retrieve the other element participating in the binary relationship using a "**Get** Element" **service** call. This is represented by Step 1008. It may be noted that the steps shown as Steps 1006 and 1008 could be combined if desired such that as each element relationship is identified, the second element in the relationship is also identified.

Detailed Description Text - DETX (111):

The use of the model in searching the Element Inventory 102 is best shown by example. Assume the model storing an element type definition for each element type is stored in the Element Inventory **Schema** 103. A particular named element, for example, element "Employee" is retrieved from the Element Inventory. This can be accomplished using the "**Get** Element" AIM DLL **service** discussed above. Next, the associated element type as indicated by the element data is retrieved from the EIS 103 using the "**Get** Element Types" AIM DLL **service**. This element type definition, which in the current example is the definition for element type "BaseTable", defines the various potential Binary Relationship Types that exist for this element. For each of these Binary Relationships Types, the metadata stored within the element is analyzed to determine if any instance of these Binary Relationship Types exists for the particular element instance. This can be done by matching relationship

indicator data stored in the element instance with a particular relationship type definition stored in the element type definition.

Detailed Description Text - DETX (113):

If desired, this process can be made recursive such that each of the identified elements is made the target of the search, and all related elements associated with the target are further identified. Using this method, the element relationships can be traversed to find all elements that are either directly, or indirectly, related to an initially-identified element. All relationship traversal is performed using the model definition. This model-based method is particularly useful because it can accommodate the situation wherein new element type definitions are added to the Element Inventory **Schema** 103, or wherein element type definitions are modified. This modification of element definitions can be accomplished without modifying any of the element management tools.

Detailed Description Text - DETX (114):

FIG. 11 is a flowchart of one embodiment of the invention utilizing a model in the manner discussed above to locate element relationships, and to locate elements related to a specified element. This algorithm is used by Element Packager 118 of the preferred embodiment, although, as stated above, the use of the model is not necessary to practice the inventive concept. The steps shown in FIG. 11 may be utilized as the process to accomplish Steps 1006 and 1008 of FIG. 10, which may be combined in the manner discussed above. As shown in FIG. 11, an unprocessed element is selected from the Current Set of elements, as shown in Step 1102. The meta-data for the current element may be obtained from the Element Inventory 102 using a call to the AIM EXE **service "Get** Element". From the element data, Element Packager 118 then locates the element type definition for the located element from the EIS 103, as illustrated in Step 1104. In the preferred embodiment, this could be performed using a call to the AIM EXE **service "Get** Element Type", which returns a specified element type definition as an **output** parameter. From this element type definition, the various potential relationship types for this element type are identified, as shown in Step 1106. For each of these relationship types, information from the current element is analyzed to determine if any instances of these potential relationship types exist, as listed in Step 1108. This determination is made using a relationship type indicator stored with each relationship instance, wherein the relationship type indicator associates a relationship type from the EIS to the relationship instance. Each relationship instance is analyzed to locate the associated element, and any newly-located element is added to the list of newly-identified elements. This is shown in step 1110. If any unprocessed elements remain in the list, this process is repeated, as represented by Decision Step 1112.

Detailed Description Text - DETX (119):

Elements of element type "Element Package" are created by the Element Packager 118 when the user selects the "Create Package" Button 708 of FIG. 7. The element name is provided by the user in Window 703. Element Packager will

read the element type definition for element type "Element Package" from the EIS using a "Get Element Type" service call. Using the retrieved definition, an instance of that element type is created in local memory having relationships of "aggregation" with the selected elements, and having a name selected by the user. Element Packager further creates an interface definition for the Element Package, wherein that interface definition is stored within the element structure. Interface definitions will be discussed further below. The newly-created element instance is written to Element Inventory 102 using the "Create Element" service call.

Detailed Description Text - DETX (122):

Before further discussing the manner in which interface attributes are used, interface definitions for Asset Elements are described. Any Asset Elements defining a code module may have a defined interface stored within the element that describes the interface for the associated code module. The interface will list one or more functions, and for each of the functions will list the input parameters required by the function, and the output parameters supplied by the completed function. An example interface specification may look like the following: Function ID1 (Firstname:string, Lastname: string), Customer_Number: short integer) Function ID2 (Firstname:string, Lastname: string), Credit_Card_Number: long integer).

Detailed Description Text - DETX (123):

This interface definition describes two functions performed by a code module associated with the element. Both identified functions include a list of input parameters shown listed between a left-most set of parenthesis. Following the input parameter list is a list of output parameters generated by the function. The above-described functions each require as input parameters strings that represent a first and last name designated as "Firstname" and "Lastname", respectively. The first function returns as an output parameter a customer number described as a short integer. The second function returns as a long integer a credit card number.

Detailed Description Text - DETX (131):

FIG. 13 is a block diagram representing the selected group of elements that models the code and data modules used to accomplish the exemplary transaction. The initial set of elements shown as "InputParameter1" 1302, "InputParameter2" 1304, and "InputParameter3" 1306 in Column 1308 is the set of elements that represent the input parameters to the transaction management system for the desired transaction.

Detailed Description Text - DETX (132):

The various code modules of the transaction management system are represented by the elements shown in Column 1316 as "Program1" 1318, and "Program2" 1320. These program elements are shown having various relationships to the input parameter elements. The various program elements in Column 1316 further generate various output parameters, which are represented and described

by the **output** parameter elements shown in Column 1322 as "OutputParameter1" 1324 and "OutputParameter2" 1326.

Detailed Description Text - DETX (133):

After the user has completed selecting the elements to include in the Element Package, the user invokes the Create Package function using the "Create Package" button 708 of FIG. 7. Element Packager automatically creates an element of type "Element Package" and assigns this element the user-specified element name, which in this case will be assumed to be "Transaction1". Element Packager creates relationships of type "aggregation" with each of the Asset Elements shown in FIG. 13. Element Packager further recognizes that the elements on the edges of the graph should be exposed. This includes those elements shown in Columns 1308 and 1322 of FIG. 13. The relationships between the Element Package element and these asset elements are assigned the Interface attribute value of "True" whereas the aggregation relationships for those elements in Column 1316 are assigned the Interface attribute value of "False". Next, Element Packager 118 creates an interface definition for the newly-created element. Element Packager may accomplish this in several way. If the exposed Asset Elements do not, themselves, store interface definitions, the Asset Element names are used to create the interface definition. That is, the Asset Elements on the left-hand side of the graph are used as the **input** parameter list, and the asset elements on the right-hand side of the graph are used as the **output** parameter list. In the current example, it will be assumed the elements in Columns 1308 and 1322 do not store any interface definitions. The interface definition created by Element Packager would be similar to the following using the Asset Element names: Function ID1 (InputParameter1:string, InputParameter2: string, InputParameter3:string), OutputParameter1:string, OutputParameter2: string)

Detailed Description Text - DETX (136):

The **input** portions of these two interface definitions would be combined to create the **input** interface definition for the Element Package, and, assuming the elements shown in Column 1322 are still included in the Element Package, these elements in Column 1322 would be utilized to generate the **output** interface definition. In this case, the interface definition for the Element Package is the same as listed above, although it is derived in a different way. Finally, if the Element Package only contained the elements in Column 1316, the entire interface definition, including both the **input and output** parameter listings, is obtained by combining the interface definitions stored in elements 1318 and 1320.

Detailed Description Text - DETX (141):

FIG. 16 is a block diagram illustrating yet another example of the creation of Element Packages, and shows how multiple Element Packages may be used to create an Element Package. Assume the **output** parameters from the first transaction represented by element "Transaction1" 1502 are to be provided to a newly-created code module. These parameters are re-formatted, then provided as **input** parameters to a transaction management system to perform a second type of

transaction. Assume further that the new code module is modeled by an asset element called "Convert1" 1602. Also, the second type of transaction is modeled by the data and asset elements included in a second Element Package called "Transaction2" 1604. The overall transaction, including transactions one and two, is modeled by an Element Package called "Transaction3". Element Packager 118 creates the interface definition for this Element Package by using the **input** portion of the interface definition for asset element "Transaction", and using the **output** portion of the interface definition for asset element "Transaction2".

Detailed Description Text - DETX (143):

FIGS. 18A and 18B, where arranged as shown in FIG. 18, are a flow diagram of the process used by Element Packager 118 to create Element Packages. In Step 1802, Element Packager retrieves information from the user interface such as the list of elements selected for inclusion in the package, the spatial relationships, and the name provided by the user for the Element Package element. In Step 1804, Element Packager retrieves the definition for the element type "Element Package" from the EIS 103 via a "**Get** Element Types" **service** call. Step 1806 illustrates the creation of a template for the newly-created Element Package element using the definition retrieved in Step 1804. Next, for each element identified by the user for inclusion in the Element Package, the element is processed according to a set of predetermined steps as shown in Step 1808. First the element definition is read from the Element Inventory, if that definition is not already located in local memory for the Element Packager. This is shown in Step 1810. Next, the element type definition is retrieved from the EIS for the element retrieved in Step 1810 if it is not already available in local memory. This is shown in Step 1812. This element type definition is used to verify that the relationship type "aggregation" is valid for the current element and that the element can be included in an Element Package, as shown in Step 1814. If the relationship type is valid for the current element, that is, the element is an Asset Element, relationship data is stored in both the newly-created Element Package element and in the related element indicating the creation of the relationship of type "aggregation". This is indicated by Step 1816. Depending on the spatial relationship of the related element as obtained from the user interface display, Element Packager determines if the related element is to be exposed. If it is, the newly-created relationship is tagged with an Interface Attribute of "True". Otherwise, the Interface Attribute is set to "False". This is illustrated in Step 1818. These steps are repeated for each identified element to be included in the Element Package, as indicated by Step 1820. As indicated by Arrow 1821, processing then continues to FIG. 18B.

Detailed Description Text - DETX (144):

Next, the interface specification is created using the elements on the left-hand side of the graph to create the **input** interface specification, and using the elements on the right-hand side of the graph to create the **output** interface specification, as shown in Steps 1822 and 1824, respectively. The interface specification will be created by combining the interface specifications of the exposed elements. For any of the elements that do not

have interface specifications, the element names themselves will be incorporated into the interface specification. The interface specification will be stored in the template for the Element Package element, as described in Step 1826. The newly-created Element Package element is written to Element Inventory 102 via a "Create Element" AIM EXE service call, and the modified element structures for those elements included in the Element Package are written back to Element Inventory 102 using an "Update Element" AIM EXE service call. This is shown in Steps 1828 and 1830, respectively.

Detailed Description Text - DETX (145):

The above description discusses the manner in which Element Packages may be used to create code wrappers and to perform impact analysis. Element Packages are also useful in systems employing component consumers and component producers as a way of making predetermined functions available to requesters. As is known in the art, a component consumer is a software process that will, in response to a user request or a request of some automated system, broadcast a request for a software module, or component, that is capable of performing a particular set of functions, and that has a predetermined interface. Requests of this nature will be received by component producers, which are software processes that will locate one or more existing software modules that meet the specified criteria, and will then return the names of these modules to the component consumer for use by the requester. A system administrator may utilize the Element Packager to define packages of elements that model groups of code and data modules, wherein each group performs a common function likely to be requested by a component consumer. The component producer can utilize the interface specifications stored in the elements to determine which packages are candidates for a request provided by a component consumer so that the name of the Element Package can be passed back to the component consumer. Thereafter, the group of code and data modules defined by the Element Package can be analyzed, either by an automated process or by human intervention, to determine the appropriateness of the match. Alternatively, if an Element Package does not already exist for the requested function, the component producer can utilize the interface specification stored in any asset elements, including other Element Packages, to determine which elements might likely be combined to provide the requested interface and functionality. This list may then be utilized by a system administrator who may then build the appropriate Element Package with all necessary elements for use by the requesting code module.

Modularization of XHTML (TM) W3C Working Draft 06 April 1999 This version: http://www.w3.org/TR/1999/xhtml-modularization-19990406

(Single HTML file [p.1] , Postscript version, PDF version, ZIP archive, or Gzip'd TAR archive) Latest version: http://www.w3.org/TR/xhtml-modularization Previous version: None. Editors: Murray Altheim, Sun Microsystems

Daniel Austin, CNET: The Computer Network Frank Boumphrey, HTML Writers Guild Sam Dooley, IBM Shane McCarron, The Open Group Ted Wugofski, Gateway 

Abstract This working draft specifies a modularization of XHTML 1.0. There are two aspects to theproposed modularization: modularization into semantic modules, and implementation of these semantic modules through a document type definition (DTD). Semantic modules provide ameans for subsetting and extending XHTML, a feature desired for extending XHTML's reach onto emerging platforms. Modularization at the DTD level improves the ability to create newcomplete DTDs from XHTML and other DTD modules.

Status of this document This document is a working draft of the W3C's HTML Working Group. This working draft may beupdated, replaced or rendered obsolete by other W3C documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work inprogress". This document is work in progress and does not imply endorsement by the W3C membership.



Modularization of XHTML(TM)Modularization of XHTML This document has been produced as part of the W3C HTML Activity. The goals of the HTMLWorking Group (members only) are discussed in the HTML Working Group charter (members only). Please send detailed comments on this document to www-html-editor@w3.org. We cannotguarantee a personal

response, but we will try when it is appropriate. Public discussion on HTML features takes place on the mailing list www-html@w3.org.

Quick Table of Contents



Full Table of Contents

modules are created according to certain conventions, as specified by the framework, to allow theircombination into semantic modules and complete, functional DTDs.

The conventions specified by the modularization framework may also be used to create newsemantic and DTD modules for other XML document types. These new semantic modules can then be used with the XHTML semantic modules to create XHTML document types thatincorporate elements from other XML document types.



Modularization of XHTML1. Introduction The modularization framework provides instructions to document authors for associating anXHTML document type with a document instance, and for verifying that the document is a valid instance of the XHTML document type associated with the document.

1.3. Modularization of XHTML The modularization of XHTML refers to the task of specifying well-defined sets of XHTMLelements that can be combined and extended by document authors, document type architects, other XML standards specifications, and application and product designers to make iteconomically feasible for content developers to deliver content on a greater number and diversity of platforms. Over the last couple of years, many specialized markets have begun looking to HTML as acontent language. There is a great movement toward using HTML across increasingly diverse computing platforms. Currently there is activity to move HTML onto mobile devices (handheldcomputers, portable phones, etc.), television devices (digital televisions, tv-based web browsers, etc.), and appliances (fixed function devices). Each of these devices has different requirementsand constraints.

Modularizing XHTML provides a means for product designers to specify which elements aresupported by a device using standard building blocks and standard methods for specifying which building blocks are used. These modules serve as "points of conformance" for the contentcommunity. The content community can now target the installed base that supports a certain collection of modules, rather than worry about the installed base that supports this permutationof XHTML elements or that permutation of XHTML elements. The use of standards is critical for modularized XHTML to be successful on a large scale. It is not economically feasible for contentdevelopers to tailor content to each and every permutation of XHTML elements. By specifying a standard, either software processes can autonomously tailor content to a device, or the devicecan automatically load the software required to process a module.

Modularization also allows for the extension of XHTML's layout and presentation capabilities,using the extensibility of XML, without breaking the XHTML standard. This development path provides a stable,

useful, and implementable framework for content developers and publishersto manage the rapid pace of technological change on the Web.

The modularization of XHTML is accomplished on two major levels: at the semantic level, and atthe document type level. Roughly speaking, the semantic level provides a conceptual approach to the modularization of XHTML, while the document type level provides DTD-level buildingblocks that allow document type designers to support the semantic modules.

1.3.1. Semantic modules An XHTML document type is defined as a set of semantic modules. A semantic module defines,in a document type, one kind of data that is semantically different from all others. Semantic modules can be combined into document types without a deep understanding of the underlyingschema that defines the modules.



1.3. Modularization of XHTMLModularization of XHTML 1.3.2. DTD modules A DTD module consists of a set of element types, a set of attribute list declarations, and a set ofcontent **model** declarations, where any of these three sets may be empty. An attribute list declaration in a DTD module may modify an element type outside the element types in themodule, and a content **model** declaration may modify an element type outside the element type set. An XML DTD is a means of describing the structure of a class of XML documents, collectivelyknown as an XML document type. XML schemas are currently represented as DTDs, as described in the XML 1.0 Recommendation [XML] [p.58] . Where possible, this document alsoallows for the potential use of other schema languages that are currently under consideration by the W3C XML Schema Working Group. (e.g. DCD, SOX, DDML, XSchema)

1.3.3. Compound document types A compound document type is an XML DTD composed from a collection of XML DTDs or DTDModules. The primary purpose of the modularization framework described in this document is to allow a DTD author to combine elements from multiple semantic modules into a compounddocument type, develop documents against that compound document type, and to validate that document against the associated compound document type. One of the most valuable benefits of XML over SGML is that XML reduces the barrier to entry forstandardization of element sets that allow communities to exchange data in an interoperable format. However, the relatively static nature of HTML as the content language for the Web hasmeant that any one of these communities have previously held out little hope that their XML document types would be able to see widespread adoption as part of Web standards. Themodularization framework allows for the dynamic incorporation of these diverse document types within the XHTML family of document types, further reducing the barriers to the incorporation ofthese domain-specific vocabularies in XHTML

documents.

1.3.4. Validation The use of well-formed, but not valid, documents is an important benefit of XML. In the processof developing a document type, however, the additional leverage provided by a validating parser for error checking is important. The same statement applies to XHTML document types withelements from multiple semantic modules.

The general problem of fragment validation - validation of XML documents with differentschemas from multiple XML Namespaces [XMLNS] [p.61] in different portions of the document - is beyond the scope of this framework. An essential feature of this framework, however, is acollection of conventions for creating, from a set of semantic modules, compound DTDs.



Modularization of XHTML1.3.2. DTD modules 1.3.5. Conformance This section introduces three notions of conformance relating to the modularization of XHTML:Document type conformance, document conformance, and browser conformance.

1.3.5.1. Document type conformance The goal of the modularization framework is to support the creation of new modules beyondthose envisioned for XHTML. To support this activity, it is the intent of this document that semantic modules form the atomic building blocks for new XHTML document types. For acompound document type to be considered an XHTML document type, it must satisfy the following properties:

It must incorporate at least one of the XHTML semantic modules. For each XHTML semantic module it uses, it must incorporate the module in its entirety.

These requirements are intended to be extremely permissive, while ensuring that documentauthors can rely on the **behavior** of a module at the semantic level.

1.3.5.2. Document conformance An XHTML document that conforms to the modularization framework must meet the following requirements:

It must specify a conforming XHTML document type. It must validate against its associated XHTML document type.

1.3.5.3. Browser conformance A browser conforming to this document shall be a conforming browser as defined in [XHTML1] [p.??] . In addition, such a browser shall support the following functionality:

1. The browser shall use the value of the xmlns attribute of the html

element to uniquelyidentify the default namespace of the document as associated with the document's XHTML

document type.

2. Terms and Definitions This section is informative. While some terms are defined in place, the following definitions are used throughout thisdocument. Familiarity with the W3C XML 1.0 Recommendation [XML] [p.58] is highly recommended. document type a class of documents sharing a common abstract structure. The ISO 8879 [SGML] [p.59]

definition is as follows: "a class of documents having similar characteristics; for example,



2. Terms and DefinitionsModularization of XHTML journal, article, technical manual, or memo. (4.102)" document **model** the effective structure and constraints of a given document type. The document modelconstitutes the abstract representation of the physical or semantic structures of a class of documents. markup **model** the markup vocabulary (ie., the gamut of element and attribute names, notations, etc.) andgrammar (ie., the prescribed use of that vocabulary) as defined by a document type definition (ie., a schema) The markup **model** is the concrete representation in markupsyntax of the document **model**, and may be defined with varying levels of strict conformity. The same document **model** may be expressed by a variety of markup models. document type definition (DTD) a formal, machine-readable expression of the XML structure and syntax rules to which adocument instance of a specific document type must conform; the schema type used in XML 1.0 to validate conformance of a document instance to its declared document type.The same markup **model** may be expressed by a variety of DTDs. reference DTD a DTD whose markup **model** represents the foundation of a complete document type. A

reference DTD provides the basis for the design of a "family" of related DTDs, such assubsets, extensions and variants. subset DTD a DTD whose document **model** is the proper subset of a reference document type, whose

conforming document instances are still valid according to the reference DTD. A subsetmay place tighter restrictions on the markup than the reference, remove elements or attributes, or both. extension DTD a DTD whose document **model** extends a reference document type (usually by the additionof element types or attributes), but generally makes no profound changes to the reference document **model** other than required to add the extension's semantic components. Anextension can also be considered a proper superset if the reference

document type is a proper subset of the extension. variant DTD a DTD whose document **model** alters (through subsetting, extension, and/or substitution)the basic data **model** of a reference document type. It is often difficult to transform without loss between instances conforming to a variant DTD and the reference DTD. fragment DTD a portion of a DTD used as a component either for the creation of a compound or variantdocument type, or for validation of a document fragment. SGML nor XML current have standardized methods for such partial validation. content **model** the declared markup structure allowed within instances of an element type. XML 1.0differentiates two types: elements containing only element content (no character data) and mixed content (elements that may contain character data optionally interspersed with childelements). The latter are characterized by a content specification beginning with the "#PCDATA" string (denoting character data).



Modularization of XHTML2. Terms and Definitions semantic module a unit of document type specification corresponding to a distinct type of content,

corresponding to a markup construct reflecting this distinct type. element type the definition of an element, that is, a container for a distinct semantic class of documentcontent. element an instance of an element type. generic identifier the name identifying the element type of an element. Also, element type name. tag descriptive markup delimiting the start and end (including its generic identifier and any

attributes) of an element. markup declaration a syntactical construct within a DTD declaring an entity or defining a markup structure.Within XML DTDs, there are four specific types: entity declaration defines the binding between a mnemonic symbol and its replacement content. element declaration constrainswhich element types may occur as descendants within an element. See also content **model**. attribute definition list declaration defines the set of attributes for a given element type, andmay also establish type constraints and default values. notation declaration defines the binding between a notation name and an external identifier referencing the format of anunparsed entity entity an entity is a logical or physical storage unit containing document content. Entities may be

composed of parseable XML markup or character data, or unparsed (ie., non-XML, possiblynon-textual) content. Entity content may be either defined entirely within the document entity ("internal entities") or external to the document entity ("external entities"). In parsed entities,the replacement text may include references to other entities. entity reference a mnemonic or numeric string used as a

reference to the content of a declared entity (eg.,

"&" for "&", "<" for "<", "©" for "Copyright 1999 Sun Microsystems, Inc.") instantiate to replace an entity reference with an instance of its declared content. parameter entity an entity whose scope of use is within the document prolog (ie., the externalsubset/DTD or internal subset). Parameter entities are disallowed within the document instance. module an abstract unit within a document **model** expressed as a DTD fragment, used toconsolidate markup declarations to increase the flexibility, modifiability, reuse and understanding of specific logical or semantic structures. modularization an implementation of a modularization **model**; the process of composing or de-composing aDTD by dividing its markup declarations into units or groups to support specific goals. Modules may or may not exist as separate file entities (ie., the physical and logicalstructures of a DTD may mirror each other, but there is no such requirement).



2. Terms and DefinitionsModularization of XHTML modularization **model** the abstract design of the document type definition (DTD) in support of the modularization

goals, such as reuse, extensibility, expressiveness, ease of documentation, code size,consistency and intuitiveness of use. It is important to note that a modularization **model** is only orthogonally related to the document **model** it describes, so that two very differentmodularization models may describe the same document type. driver a generally short file used to declare and instantiate the modules of a DTD. A good rule of

thumb is that a DTD driver contains no markup declarations that comprise any part of thedocument **model** itself. parent document type A parent document type of a compound document is the document type of the root element. compound document A compound document is a document that uses more than one XML Namespace.

Compound documents may be defined as documents that contain elements or attributesfrom multiple document types. A module is a collection of elements or attributes. A profile is metadata about an XML document type and possible related technologies, such as scripting languages and style-sheets. A browser can support one or many profiles. Thepurpose of profiles is to provide content developers with machine-readable information about features that can be expected from a particular browser. For example, a Televisionprofile would include a DTD for Television sets and describe technologies that can be used (e.g. scripts and style-sheets).

3. XHTML Semantic Modules This section is normative. This section

specifies the contents of the XHTML semantic modules. Semantic modules areabstract definitions of collections of elements, attributes, and their content models. These semantic modules can be mapped onto any appropriate specification mechanism. The nextsection, for example, maps these modules onto DTDs as described in [XML].

Throughout this section, some elements are marked as only being available when thetransitional aspects of XHTML are supported. Transitional elements, attributes, and behaviors are defined in HTML 4. These transitional aspects continue to be defined by XHTML, butXHTML does not require their support in all conforming implementations.

Content developers and device designers should view this section as a guide to the definition ofthe functionality provided by the various XHTML-defined modules. When developing documents or defining a profile for a class of documents, content developers can determine which of thesemodules are essential for conveying their message. When designing clients, device designers should develop their device profiles by choosing from among the abstract modules defined here.



Modularization of XHTML3. XHTML Semantic Modules 3.1. Applet Module The Applet Module provides elements for including external applets. Specifically, the AppletModule supports:

applet param

3.2. Block Modules Block modules define "block level" elements and their attributes. Block level elements are thosethat cause a break in the rendered output when they are encountered in a document.

3.2.1. Block Phrasal Module This module defines block level elements that have special "phrasal" rendering characteristics -they transform the enclosed content in a special, specific way to assist in its interpretation by the user. Elements included in the Block Phrasal Module are:

address blockquote pre h1 h2 h3 h4 h5 h6

3.2.2. Block Presentational Module This module defines block level elements that are used strictly to improve the presentation of adocument. Elements included in the Block Presentation Module are:

center* hr

Items marked with an asterisk are only available when the Transitional aspects of XHTML are enabled.

3.2.3. Block Structural Module This module defines block level elements to help control the structure of their enclosed content.Elements included are:

- 13 -

3.1. Applet ModuleModularization of XHTML div p

3.3. Inline modules Inline modules defined elements and their attributes that, when used in a document, effect theircontents but do not cause a break in the rendered output.

3.3.1. Inline Phrasal Module This module defines inline level elements that have special "phrasal" rendering characteristics -they transform the enclosed content in a special, specific way to assist in its interpretation by the user. Elements included in the Inline Phrasal Module are:

abbr acronym cite code dfn em kbd q samp strong var

3.3.2. Inline Presentational Module This module defines inline level elements that are used strictly to improve the presentation of adocument. Elements included in the Inline Presentation Module are:

b basefont* big font* i s* small strike* sub sup tt u*

- 14 -

Modularization of XHTML3.3. Inline modules Items marked with an asterisk are only available when the Transitional aspects of XHTML are enabled. 3.3.3. Inline Structural Module This module defines inline level elements to help control the structure of their enclosed content.Elements included are:

bdo br del ins span

3.4. Linking Module The Linking Module provides elements that are used for linking an HTML document to otherdocuments or resources. Specifically, the Linking Module supports:

a base link

3.5. List Module As its name suggests, the List Module provides list-oriented elements. Specifically, the ListModule supports:

dir* dl dt dd ol ul li menu*

Items marked with an asterisk are only available when the Transitional aspects of XHTML are enabled.

3.4. Linking ModuleModularization of XHTML 3.6. HTML 3.2 Forms Module The HTML 3.2 Forms Module provides the forms features found in HTML 3.2. Specifically, theHTML 3.2 Forms Module supports:

form input select option textarea

3.7. HTML 4.0 Forms Module The HTML 4.0 Forms Module provides all of the forms features found in HTML 4.0. Specifically,the HTML 4.0 Forms Module supports:

button fieldset form input label legend optgroup option select textarea

The HTML 4.0 Forms Module is a superset of the HTML 3.2 Forms Module.

3.8. HTML 3.2 Table Module The HTML 3.2 Table Module provides table-related elements, but only in a limited form.Specifically, the HTML 3.2 Table Module supports:

caption table td th tr



Modularization of XHTML3.6. HTML 3.2 Forms Module 3.9. HTML 4.0 Table Module As its name suggests, the HTML 4.0 Table Module provides table-related elements, with the fullcapabilities of HTML 4.0. Specifically, the HTML 4.0 Table Module supports:

caption col colgroup table tbody td tfoot th thead tr

This module is a proper superset of the HTML 3.2 Table Module.

3.10. Image Module The Image Module provides basic image embedding, and may be used in someimplementations independently of client side image maps. The Image Module supports:

img 3.11. Image Map Module The Image Map Module provides elements for images and client side image maps. Specifically,the Image Map Module supports:

area map

The Image Map Module is typically used in conjunction with the image [p.17] and linking [p.15] modules

3.12. Object Module The Object Module provides elements for general-purpose object inclusion. Specifically, theObject Module supports:

object param



3.9. HTML 4.0 Table ModuleModularization of XHTML 3.13. Frames Module As its name suggests, the Frames Module provides frame-related elements. Specifically, theFrames Module supports:

frameset frame iframe noframes

The elements in the Frame Module are included in the HTML 4.0 Frameset document type.

3.14. Intrinsic Events Intrinsic events are attributes that are used in conjunction with elements that can have specificactions occur when certain events are performed by the user. Attributes included in this module are:

onclick ondblclick onmousedown onmouseup onmouseover onmousemove onmouseout onkeypress onkeydown onkeyup

3.15. Metainformation Module The Metainformation Module defines elements that are typically used within the declarativeportion of a document (in XHTML within the head element). This module includes the following elements:

meta title

3.16. Scripting Module The Scripting Module defines elements that are used to contain information pertaining toexecutable scripts or the lack of support for executable scripts. Elements included in this module are:



Modularization of XHTML3.13. Frames Module noscript script

3.17. Stylesheet Module The Stylesheet Module defines style sheet handling. The module includes:

style

3.18. Structure Module The Structure Module defines the major structural elements for XHTML. These elementseffectively act as the basis for the content **model** of HTML (although the content **model** itself is defined in other, DTD-specific modules). The elements included in this module are:

body head html

4. XHTML DTD Modules This section is normative. This section specifies the XHTML DTD modules, the type declarations found in each module,and the file and parameter entity naming conventions used throughout the DTD.

4.1. Implementing Document **Model** Modules in the DTD Partitioning of the document **model** occurs at the semantic module level. This partitioning isimplemented in the markup **model** by two primary methods: parameterization, the use of parameter entities as reusable strings, and modularization, the creation of DTD fragments called modules.

4.1.1. Parameterization This specification classifies parameter entities into six categories and names them consistentlyusing the following suffixes:

.mod parameter entities use the suffix

.mod when they are used to represent a DTD module (acollection of element classes). In this specification, each module is an atomic unit and may

be represented as a separate file entity. .module parameter entities use the suffix .module when they are used to control the inclusion of aDTD module by containing either of the conditional section keywords

INCLUDE or IGNORE.





.content when they are used to represent the contentmodel of an element type.

.class parameter entities use the suffix

.class when they are used to represent elements of thesame class.

.mix parameter entities use the suffix

.mix when they are used to represent a collection ofelement types from different classes.

.attrib parameter entities use the suffix

.attrib when they are used to represent a group oftokens representing one or more complete attribute specifications within an ATTLIST

declaration. For example, in HTML 4.0, the %block; parameter entity is defined to represent theheterogenous collection of element types that are block-level elements. In this specification, the corollary parameter entity is %Block.mix;.

4.1.2. Modularization DTD modules are often used to encompass the markup declarations of a specific semanticcomponent or "feature", from higher-level document features like tables and forms, to lower-level components such as specific elements or element groups. Modules can even contain modules,creating a hierarchical structure mirroring the document **model**. Note that modules are not always implemented as separate file entities, and modular DTDs can be easily normalized intosingle file versions for more efficient distribution over the Web.

The relationship between document **model** components and how they are implemented inmarkup as modules, entities and files (i.e., the granularity of the parameterization or modularization, how the markup **model** is structured and stored as separate entities, etc.) is notnecessarily direct, as design style and implementation issues properly play a part. Higher-level modules are sometimes delivered as individual file entities to facilitate portability and reusability.To promote interoperability, the XHTML DTD design considers each module as atomic, with the notion that implementations should support the semantics of an entire module without further subdivision.

While the notion of "plug and play" with DTD modules is very attractive, in practice this is notquite so simple. Complex document models often resort to extensive parameterization of semantic modules to facilitate understanding, markup reuse, extensibility, and maintenance. Theresultant modules may have have many interdependencies, and may require a fair amount of "rewiring" when adding or removing a DTD module. In light of this, a compromise must be madebetween markup flexibility, complexity of the DTD, and ease of maintainability.

The XHTML DTD attempts to ameliorate this by localizing many of the more "global" parameterentities to several modules that are declared early in the DTD. These are labelled common modules, and include declarations for common names, attributes, parameter and character entities.



Modularization of XHTML4.1.2. Modularization XHTML elements are classified into the following categories: structural element types element types that create the overall structure of an XHTML document. block element types element types that should cause a line break. inline element types element types that are displayed inline to an existing block. phrasal element types element types that specify a

domain-relevant connotation presentational element types element types that indicate a desire on the part of the author for a specific rendering effect. special case (or "feature") element types element types that provide XHTML with special features, such as linking, forms, etc.

Following is a module-by-module catalog of element types and parameter entities declared inthe three XHTML DTDs, what constitutes in effect the XHTML namespace. More detailed DTD Modularization **Interface** ('DMI') documentation is also available for each ofthe three XHTML 1.0 DTDS:

XHTML 1.0 Strict DMI XHTML 1.0 Transitional DMI XHTML 1.0 Frameset DMI

4.2. Common Declarations 4.2.1. Names (XHTML1-names.mod) The XHTML1-names.mod DTD module defines the following common names, many of theseimported from other specifications and standards.

%ContentType; media type, as per [RFC2045] %ContentTypes; a comma-separated list of media types, as per [RFC2045] %Charset; a character encoding, as per [RFC2045] %Charsets; a space-separated list of character encodings, as per [RFC2045] %Datetime; date and time information, ISO date format %Character; a single character from [ISO10646] %LanguageCode; a language code, as per [RFC1766]



4.2. Common DeclarationsModularization of XHTML %LinkTypes; a space-separated list of link types %MediaDesc; a single or comma-separated list of media descriptors %Number; one or more digits %URI; a Uniform Resource Identifier, see [URI] %URIs; a space-separated list of Uniform Resource Identifiers, see [URI] %Script; a script expression %StyleSheet; style sheet data %Text; simple text %Length; the length defined in the HTML strict DTD for cellpadding and cellspacing % MultiLength; pixel, percentage, or relative %MultiLengths; a comma-separated list of MultiLength %Pixels; integer representing length in pixels %FrameTarget; render in this frame %Color; a color using sRGB

4.2.2. Attributes (XHTML1-attribs.mod) The XHTML1-attribs.mod DTD modules defines the following common attributes: %Core.attrib; defines the attributes

id, class, style, and title %I18n.attrib;

defines the internationalization (i18n) attributes lang, xml:lang and dir %Events.attrib; in this module, %Events.attrib; is declared as an empty string, should the Eventsmodule not already have been instantiated. %Common.attrib; combines

%Core.attrib;, %I18n.attrib;, and %Events.attrib; %Align.attrib;

in this module, %Align.attrib; is declared as an empty string, as a default in the StrictDTD.



Modularization of XHTML4.2.2. Attributes (XHTML1-attribs.mod) % XLink.attribs; a conditional section keyword controlling declaration of additional XLink attributes on the

aelement type

%Alink.attrib; additional XLink attributes on the

a element type

4.2.3. Transitional Attributes (XHTML1-attribs-t.mod) The XHTML1-attribs-t.mod DTD module defines the common attributes associated with theHTML 4.0 Transitional DTD:

%Core.attrib; defines the attributes

id, class, style, and title %I18n.attrib;

defines the internationalization (i18n) attributes lang, xml:lang and dir %Common.attrib; combines %Core.attrib;, %I18n.attrib;, and % Events.attrib; %Align.attrib; horizontal text alignment % IAlign.attrib; horizontal and vertical text alignment %XLink.attribs; a conditional section keyword controlling declaration of additional XLink attributes on the aelement type %Alink.attrib; additional XLink attributes on the

a element type

4.2.4. Strict Document **Model** (XHTML1-**model**.mod) The XHTML1-**model** DTD module declares parameter entities describing the structure of theXHTML Strict document **model**. It provides an effective implementation of the document **model** occurring within body in one location, simplifying modification. It's also a good place to gain anunderstanding of the structures of the DTD.

NOTE: because the #PCDATA token must occur first in a mixed content **model** in XML, it is notincluded in any of the following parameter entities, and is declared explicitly on each element where it is to occur. %Misc.class; these elements are neither block nor inline, and can essentially be used anywhere in the

document body %Inlstruct.class; the class of inline structural

element types %Inlpres.class; the class of inline presentational element types

- 23 -

4.2.3. Transitional Attributes (XHTML1-attribs-t.mod)Modularization of XHTML %Inlphras.class; the class of inline phrasal element types % Inlspecial.class; the class of special inline element types % Formctrl.class; the class of form control element types % Inline.class; includes all inline elements, used as a component in mixes %Inline.mix; includes all inline elements, including

%Misc.class; %Inline-noa.class;

includes all non-anchor inlines, used as a component in mixes % Inline-noa.mix; includes all non-anchor inlines %Heading.class; the class of heading element types h1 to h6 %List.class; the class of list element types %Blkstruct.class; the class of block structural element types %Blkpres.class; the class of block presentational element types %Blkphras.class; the class of block phrasal element types %Blkform.class; the class of block-level form element types % Blkspecial.class; the class of special block-level element types % Block.class; includes all block elements, used as an component in mixes %Block.mix; includes all block elements plus %Misc.class; % Block-noform.class; includes all non-form block elements, used as a component in mixes %Block-noform.mix; includes all non-form block elements, plus %Misc.class; %Flow.mix; includes all text content, block and inline

4.2.5. Transitional Document **Model** (XHTML1-**model**-t.mod) The XHTML1-**model** DTD module declares parameter entities describing the structure of theXHTML Transitional document **model**. It provides an effective implementation of the document **model** occurring within body in one location, simplifying modification. It's also a good place togain an understanding of the structures of the DTD.

- 24 -

Modularization of XHTML4.2.5. Transitional Document **Model** (XHTML1-**model**-t.mod) NOTE: because the #PCDATA token must occur first in a mixed content **model** in XML, it is notincluded in any of the following parameter entities, and is declared explicitly on each element where it is to occur. %Misc.class; these elements are neither block nor inline, and can essentially be used anywhere in the

document body %Inlstruct.class; the class of inline structural element types %Inlpres.class; the class of inline presentational element types %Inlphras.class; the class of inline phrasal element types %Inlspecial.class; the class of special inline element types %

Inlspecial.class; [description] %Formctrl.class; the class of form control element types %Inline.class; includes all inline elements, used as a component in mixes %Inline.mix; includes all inline elements, including %Misc.class; %Inline-noa.class; includes all non-anchor inlines, used as a component in mixes %Inline-noa.mix; includes all non-anchor inlines %Heading.class; the class of heading element types h1 to h6 %List.class; the class of list element types % Blkstruct.class; the class of block structural element types % Blkpres.class; the class of block presentational element types % Blkphras.class; the class of block phrasal element types % Blkform.class; the class of block-level form element types % Blkspecial.class; the class of special block-level element types % Blkspecial.class; the class of special block-level element types % Block.class; includes all block elements, used as an component in mixes



4.2.5. Transitional Document **Model** (XHTML1-**model**-t.mod)Modularization of XHTML %Block.mix; includes all block elements plus

%Misc.class; %Block-noform.class;

includes all non-form block elements, used as a component in mixes % Block-noform.mix; includes all non-form block elements, plus % Misc.class; %Flow.mix; includes all text content, block and inline

4.2.6. Intrinsic Event Attributes (XHTML1-events.mod) The XHTML1-events.mod DTD module defines the intrinsic event attributes specified in HTML 4.0.

ATTLIST a additional event-related attributes for the

a element type

%Events.attrib; defines the intrinsic events such as

onclick and onmouseout

4.2.7. Character Entities (XHTML1-charent.mod) This module acts as a wrapper for declaring the three character entity sets based on ISO Latin 1and other special symbols used in HTML, such as

<, , Ä, etc.

%XHTML1.ents; a conditional section keyword which can be set to

IGNORE during normalization of the DTDto avoid having the character entity declarations included in the resultant DTD (this should

be done in internal subset of the dummy document used for normalization, rather thanexplicitly editing this module) %XHTML1-lat1; the extended ISO Latin 1 set of character entities, the same as in HTML 4.0 DTD. %XHTML1-symbol; symbol characters for XHTML, including mathematical and Greek characters %XHTML1-special; special characters for XHTML, including typographic symbols. (this set now includes the

new € symbol)

4.3. Document Structure Element Types 4.3.1. Normal Document Structure (XHTML1-struct.mod) The XHTML1-struct.mod DTD module defines element types that support the general structureof an XHTML document, as apart from the content's structure.



Modularization of XHTML4.3. Document Structure Element Types ELEMENT head the

head element type and its attributes %Head.content;

the content **model** for the head element type %Head-opts.mix; the optional, repeatable element types that can appear in the head of a document ELEMENT body the body element type and its attributes ATTLIST body additional Transitional attributes on body (in a % XHTML.Transitional; conditionalsection) %Body.content; the content **model** for the

body element type ELEMENT html

the html element type and its attributes, the document element for XHTML. %Html.content; the content **model** for the html element type % Version.attrib; the Formal Public Identifier (FPI) for this DTD. This parameter entity is a historical legacy of

past HTML DTDs, and is preserved in case any application software uses it. It is also theonly place within the DTD (excluding comments) that the DTD's FPI is declared. This declares the attribute specification containing the value as declared in the DTD driver. % Profile.attrib; the declared profile identifier for this DTD. FIXME This should be changed in the DTD to be named as a namespace, not profile identifier.

4.3.2. Frames (XHTML1-frames.mod) The XHTML1-frames.mod DTD module defines element types that provide frame functionality. ELEMENT frameset the

frameset element type and its attributes %Frameset.content;

the content **model** for the frameset element type ELEMENT frame the frame element type and its attributes %Frame.content; the content **model** for the frame element type ELEMENT iframe the iframe element type and its attributes %Iframe.content; the content **model** for the frame element type ELEMENT noframes the noframe element type and its attributes



4.3.2. Frames (XHTML1-frames.mod)Modularization of XHTML % Noframes.content; the content **model** for the

noframes element type ATTLIST a

additional frame-related attributes on the a element type % Html.content; redeclares the content **model** of the html element type

4.3.3. Lists (XHTML1-list.mod) The XHTML1-lists.mod DTD module defines elements that provide list functionality. ELEMENT dl the

dl (definition list) element type and its attributes %Dl.content;

the content **model** for the dl element type ELEMENT dt the dl (definition term) element type and its attributes %Dt.content; the content **model** for the dt element type ELEMENT dd the dl (definition description) element type and its attributes %Dd.content; the content **model** for the dd element type ELEMENT ol the dl (ordered list) element type and its attributes %Ol.content; the content **model** for the ol element type %OlStyle; ordered lists (ol) numbering style ELEMENT ul the dl (unordered list) element type and its attributes % Ul.content; the content **model** for the ul element type %UlStyle; unordered list (ul) bullet styles ELEMENT li the dl (list item) element type and its attributes %Li.content; the content **model** for the li element type ELEMENT dir declares the Transitional element type dir (deprecated in HTML 4.0) %Dir.content; the content **model** for the dir element type ELEMENT menu declares the Transitional element type menu (deprecated in HTML 4.0)



Modularization of XHTML4.3.3. Lists (XHTML1-list.mod) %Menu.content; the content **model** for the

menu element type

4.4. Block Element Types 4.4.1. Block Structural (XHTML1-blkstruct.mod) The XHTML1-blkstruct.mod DTD module defines element types that provide block-level structure.

ELEMENT div the

div element type and its attributes %Div.content;

the content **model** for the div element type ELEMENT p the p element type and its attributes %P.content; the content **model** for the p element type

4.4.2. Block Phrasal (XHTML1-blkphras.mod) The XHTML1-blkphras.mod DTD module defines element types that provide block-levelsemantic features.

ELEMENT address the

address element type and its attributes %Address.content;

the content **model** for the address element type ELEMENT blockquote the blockquote element type and its attributes %Blockquote.content; the content **model** for the blockquote element type ELEMENT pre the pre element type and its attributes %Pre.content; the content **model** for the pre element type ELEMENT h1 the h1 element type and its attributes ELEMENT h2 the h2 element type and its attributes ELEMENT h3 the h3 element type and its attributes ELEMENT h4 the h4 element type and its attributes



4.4. Block Element TypesModularization of XHTML ELEMENT h5 the

h5 element type and its attributes ELEMENT h6

the h6 element type and its attributes %Heading.content; the content **model** used by all heading elements (h1-h6)

4.4.3. Block Presentational (XHTML1-blkpres.mod) The XHTML1-blkpres.mod DTD module defines element types that provide block-levelpresentational features.

ELEMENT hr the

hr element type and its attributes %Hr.content;

the content **model** for the hr element type ELEMENT center the center element type and its attributes %Center.content; the content **model** for the center element type

4.5. Inline Element Types 4.5.1. Inline Structural (XHTML1-inlstruct.mod) The XHTML1-inlstruct.mod DTD module defines element types that provide inline structural features.

ELEMENT bdo the

bdo element type and its attributes %Bdo.content;

the content **model** for the bdo element type ELEMENT br the br element type and its attributes %Br.content; the content **model** for the br element type ELEMENT del the del element type and its attributes % Del.content; the content **model** for the del element type ELEMENT ins the ins element type and its attributes %Ins.content; the content **model** for the ins element type



Modularization of XHTML4.5. Inline Element Types ELEMENT span the

span element type and its attributes %Span.content;

the content **model** for the span element type

4.5.2. Inline Phrasal (XHTML1-inlphras.mod) The XHTML1-inlphras.mod DTD module defines element types that provide inline phrasal features.

ELEMENT abbr the

abbr element type and its attributes %Abbr.content;

the content **model** for the abbr element type ELEMENT acronym the acronym element type and its attributes %Acronym.content; the content **model** for the acronym element type ELEMENT cite the cite element type and its attributes %Cite.content; the content **model** for the cite element type ELEMENT code the code element type and its attributes % Code.content; the content **model** for the code element type ELEMENT dfn the dfn element type and its attributes %Dfn.content; the content **model** for the dfn element type ELEMENT em the em element type and its attributes %Em.content; the content **model** for the em element type ELEMENT kbd the kbd element type and its attributes %Kbd.content; the content **model** for the kbd element type ELEMENT q the q element type and its attributes %Q.content; the content **model** for the q element type ELEMENT samp the samp element type and its attributes % Samp.content; the content **model** for the samp element type



4.5.2. Inline Phrasal (XHTML1-inlphras.mod)Modularization of XHTML ELEMENT strong the

strong element type and its attributes %Strong.content;

the content **model** for the strong element type ELEMENT var the var element type and its attributes %Var.content; the content **model** for the var element type

4.5.3. Inline Presentational (XHTML1-inlpres.mod) The XHTML1-inlpres.mod DTD module defines element types that provide inline presentational features.

ELEMENT b the

b element type and its attributes %B.content;

the content **model** for the b element type ELEMENT big the var element type and its attributes %Big.content; the content **model** for the big element type ELEMENT i the i element type and its attributes % I.content; the content **model** for the i element type ELEMENT small the small element type and its attributes %Small.content; the content **model** for the small element type ELEMENT sub the sub element type and its attributes %Sub.content; the content **model** for the sub element type ELEMENT sup the sup element type and its attributes % Sup.content; the content **model** for the sup element type ELEMENT tt the tt element type and its attributes %Tt.content; the content **model** for the tt element type ELEMENT basefont the basefont element type and its attributes %Basefont.content; the content **model** for the basefont element type





font element type and its attributes %Font.content;

the content **model** for the font element type ELEMENT s the s element type and its attributes %S.content; the content **model** for the s element type ELEMENT strike the strike element type and its attributes %Strike.content; the content **model** for the strike element type ELEMENT u the u element type and its attributes %U.content; the content **model** for the u element type

4.6. Special Case (or Feature) Element Types 4.6.1. Meta Information (XHTML1-meta.mod) The XHTML1-meta.mod DTD module defines element types that provide document metainformation. All occur in

head.

ELEMENT title the

title element type and its attributes %Title.content;

the content **model** for the title element type ELEMENT meta the meta element type and its attributes %Meta.content; the content **model** for the meta element type

4.6.2. Linking (XHTML1-linking.mod) The XHTML1-linking.mod DTD module defines element types that provide inline link and linkreference features.

ELEMENT a the

a element type and its attributes %A.content;

the content **model** for the a inline link element type ELEMENT base the base element type and its attributes



4.6. Special Case (or Feature) Element TypesModularization of XHTML % Base.content; the content **model** for the

base element type ELEMENT link

the link element type and its attributes %Link.content; the content **model** for the link element type %Shape; enumeration of shape values for client-side image maps %Coords; comma-separated list of vector coordinates for client-side image maps

4.6.3. Images (XHTML1-image.mod) The XHTML1-image.mod DTD module defines element types that provide inline image support. ELEMENT img the

img element type and its attributes %Img.content;

the content **model** for the img element type

4.6.4. Client-side Image Maps (XHTML1-csismap.mod) The XHTML1-csimap.mod DTD module defines element types that provide client-side imagemap support:

ELEMENT map the

map element type and its attributes %Map.content;

the content **model** for the map element type ELEMENT area the area element type and its attributes %Area.content; the content **model** for the area element type ATTLIST a additional client-side image map attributes on

a

4.6.5. Objects (XHTML1-object.mod) The XHTML1-object.mod DTD module defines elements that support generic embedded objects.

ELEMENT object the

object element type and its 3,417 attributes



Modularization of XHTML4.6.3. Images (XHTML1-image.mod) % Object.content; the content **model** for the

object element type ELEMENT param

the param element type and its attributes %Param.content; the content **model** for the param element type

4.6.6. Applets (XHTML1-applet.mod) The XHTML1-applet.mod DTD module defines element types that provide support for Java applets.

ELEMENT applet the

applet element type and its attributes %Applet.content;

the content **model** for the applet element type ELEMENT param the param element type and its attributes %Param.content; the content **model** for the param element type %Param.local.module; if the Object module is not included, this conditional section keyword should be declared as INCLUDE to declare the param element and its attributes

4.6.7. Scripting (XHTML1-script.mod) The XHTML1-script.mod DTD module defines element types that provide scripting support. ELEMENT script the

script element type and its attributes %Script.content;

the content **model** for the script element type ELEMENT noscript the noscript element type and its attributes %Noscript.content; the content **model** for the noscript element type

4.6.8. Stylesheets (XHTML1-style.mod) The XHTML1-style.mod DTD module defines element types that provide stylesheet support. ELEMENT style the

style element type and its attributes %Style.content;

the content **model** for the style element type

4.6.6. Applets (XHTML1-applet.mod)Modularization of XHTML 4.6.9. HTML 3.2 Tables (XHTML1-table32.mod) The XHTML1-table32.mod DTD module defines elements that provide support for display ofHTML 3.2 tables.

ELEMENT table the

table element type and its attributes %Table.content;

the content **model** for the table element type ELEMENT caption the caption element type and its attributes %Caption.content; the content **model** for the caption element type %CaptionAlign; specifies alignment of the caption relative to the table ELEMENT tr the tr element type and its attributes %Tr.content; the content **model** for the tr element type ELEMENT th the th element type and its attributes %Th.content; the content **model** for the th element type ELEMENT td the td element type and its attributes %Td.content; the content **model** for the td element type %TAlign; horizontal placement of table relative to document %CellHAlign.attrib; horizontal alignment attributes for cell contents %CellVAlign.attrib; vertical alignment attributes for cell contents

4.6.10. HTML 4.0 Tables (XHTML1-table.mod) The XHTML1-table.mod DTD module defines elements that provide support for display of HTML4.0 tables.

ELEMENT table the

table element type and its attributes %Table.content;

the content **model** for the table element type ELEMENT caption the caption element type and its attributes



Modularization of XHTML4.6.9. HTML 3.2 Tables (XHTML1-table32.mod) % Caption.content; the content **model** for the

caption element type %CaptionAlign;

specifies alignment of the caption relative to the table ELEMENT thead the thead element type and its attributes %Thead.content; the content **model** for the thead element type ELEMENT tfoot the tfoot element type and its attributes %Tfoot.content; the content **model** for the tfoot element type ELEMENT tbody the tbody element type and its attributes %Tbody.content; the content **model** for the tbody element type ELEMENT colgroup the colgroup element type and its attributes % Colgroup.content; the content **model** for the colgroup element type ELEMENT col the col element type and its attributes %Col.content; the content **model** for the col element type ELEMENT tr the tr element type

and its attributes %Tr.content; the content **model** for the tr element type ELEMENT th the th element type and its attributes %Th.content; the content **model** for the th element type ELEMENT td the td element type and its attributes %Td.content; the content **model** for the td element type %TFrame; the

frame attribute values specify which parts of the frame around the table should berendered

%TRules; specifies which rules to draw between cells %TAlign; horizontal placement of table relative to document % CellHAlign.attrib; horizontal alignment attributes for cell contents



4.6.10. HTML 4.0 Tables (XHTML1-table.mod)Modularization of XHTML % CellVAlign.attrib; vertical alignment attributes for cell contents % Scope; describes the scope of cells covered by header cells; scope is simpler than the axes

attribute for common tables

4.6.11. HTML 3.2 Forms (XHTML1-form32.mod) The XHTML1-form32.mod DTD module defines element types that provide HTML 3.2 level form support.

ELEMENT form the

form element type and its attributes %Form.content;

the content **model** for the form element type ELEMENT input the input element type and its attributes %InputType.class; changes to the input element's type attribute values %Input.content; the content **model** for the input element type ELEMENT select the select element type and its attributes %Select.content; the content **model** for the select element type ELEMENT option the option element type and its attributes %Option.content; the content **model** for the option element type ELEMENT textarea the textarea element type and its attributes % Textarea.content; the content **model** for the textarea element type

4.6.12. HTML 4.0 Forms (XHTML1-form.mod) The XHTML1-form.mod DTD module defines element types that provide HTML 4.0 level form support.

ELEMENT form the

form element type and its attributes %Form.content;

the content **model** for the form element type ELEMENT label the label

element type and its attributes



Modularization of XHTML4.6.11. HTML 3.2 Forms (XHTML1-form32.mod) % Label.content; the content **model** for the

label element type ELEMENT input

the input element type and its attributes %InputType.class; changes to the input element's type attribute values %Input.content; the content **model** for the input element type ELEMENT select the select element type and its attributes %Select.content; the content **model** for the select element type ELEMENT optgroup the optgroup element type and its attributes %Optgroup.content; the content **model** for the optgroup element type ELEMENT option the option element type and its attributes %Option.content; the content **model** for the option element type ELEMENT textarea the textarea element type and its attributes % Textarea.content; the content **model** for the textarea element type ELEMENT fieldset the fieldset element type and its attributes % Fieldset.content; the content **model** for the fieldset element type ELEMENT legend the legend element type and its attributes % Legend.content; the content **model** for the legend element type % LegendAlign.attrib; values for legend alignment ELEMENT button the button element type and its attributes %Button.content; the content **model** for the button element type

5. Specifying XHTML Profiles This section is informative. Modularization of XHTML provides building blocks for language, application, and platformdevelopers. This document recognizes that, in addition to semantic modules, there exists the notion of profiles.



5. Specifying XHTML ProfilesModularization of XHTML In addition to the definition of modules and profiles, a framework is necessary for two or moredevices to negotiate when there is a mismatch in profiles supported (or required by content be transmitted). This section focuses on modules and the process in which modules may be combined into aprofile (using the document type definition language).

5.1. Framework for Content Negotiation Content negotiation is the process in which two or more devices select content (or a set ofcontent) to transmit from a sender to a receiver. This selection is based upon:

the capabilities, or features, that the content requires, the capabilities of the sending device, and the capabilities of the receiving device.

In the general case, a sender will transmit content to a receiver along a transmission path; this isillustrated in Figure 5-1 [p.40] .

Sometimes there will be a mismatch of capabilities between the sender, the content to send, andthe potential receiver of the content; content negotiation is the reconciliation of these mismatches. As identified in [FRMWRK] [p.62] , content negotiation covers three elements:

1. expressing the capabilities of the sender and the content to be transmitted 2. expressing the capabilities of the receiver, and 3. the protocol by which the capabilities are exchanged.

These elements are consistent with a broadcast scenario in which there is a uni-directional linkbetween the originator of the content and the destination of the content, the client device; this is illustrated in Figure 5-2 [p.41] :



 In Figure 5-2 [p.41] , the receiver does not have a direct connection to the sender, rather a proxyserves to represent the capabilities of the receiver. Where the proxy also has reformatting, transformation, or translation capability, the proxy's capabilities may also be represented. As [FRMWRK] [p.62] indicates, negotiation between the sender and the receiver (which may becarried out in abstentia by a proxy) consists of a series of negotiation exchanges that proceeds until either party (the sender or receiver) determines that a specific data file or content betransmitted. As Figure 5-2 [p.41] illustrates, not every network architecture will support this process in a general sense, but an implementation of content negotiation should be a subset ofthis framework. The W3C Note on the CC/PP exchange protocol [CCPP] [p.60] supports a subset of this framework that describes the capabilities and preferences associated with usersand user agents accessing the World Wide Web.

The HTML Working Group will identify how the XHTML profiles can be specified (in terms ofexpressiveness and syntax) and the requirements placed upon an appropriate negotiation protocol.

5.2. Expressing Profiles A profile is a collection of XHTML modules (particular to an application domain), specification ofstyle sheet and scripting support, preferences, and other user agent and/or document properties. Typically, a key component of the profile is a document type definition (DTD). This DTD may berepresented using a URL or the profile may include the DTD itself. The problem with this approach is that if a URL is used, the actual DTD may not be available (due to networkunavailability, perhaps), including the

entire DTD with every document may require too much bandwidth, and a device that wishes to process the profile may not have the resources orcapabilities to process a DTD.

Therefore, there is a need to define a more general purpose solution for specifying a profile'sfeatures or feature sets (the elements, attributes, and modules contained in a profile). As predicted in [FRMWRK] [p.62] , this solution should provide:

a vocabulary for designating a profile's features and feature sets, and an extensible framework to allow adoption of new features.



5.2. Expressing ProfilesModularization of XHTML In some cases, multiple representations of the same data may be available (for example, thiscan be realized through CSS media rules or nested object elements). These multiple representations raise additional requirements, also predicted by [FRMWRK] [p.62] , including:

a means for indicating preferences, and a means for capturing dependencies between feature values.

Functionally, profiles must be able to express:

the appropriate document type definition or definitions, the XHTML modules that are contained, and elements and attributes outside of a module that are contained or excluded.

Varying syntaxes may be used for representing these profiles: CC/PP [CCPP] [p.60] isspecifying a syntax based on RDF [RDF], [p.60] [SYNTAX] [p.62] proposes a syntax built upon mathematical relationships, and [SYNURL] [p.62] provides extensions to [SYNTAX] [p.62] fordereferencing (essentially abbreviating) the proposed syntax with URLs.

5.3. Syntax for Representing XHTML Modules The syntax chosen for representing an XHTML Modules should satisfy the following functional requirements:

Receiver and Sender Capabilities

the syntax may identify a collection of supported XHTML Modules through a DTD the syntax may identify XHTML Modules at the "named" level; i.e., modules can bespecified without specific reference to element and attribute support

the syntax may identify elements and attributes that are not supported in an XHTMLModule that is supported the syntax may identify

elements and attributes that are supported outside of anXHTML Module the syntax may identify XHTML Module preferences the syntax may identify attribute value support (such as scripting languages and mediatypes)

the syntax may identify attribute value preferences Additional Sender Capabilities

the syntax may identify XHTML Module transformation support the syntax may identify element and attribute transformation support the syntax may identify transformation preferences Content Capabilities (Features)

the syntax may identify a collection of required XHTML Modules through a DTD the syntax may identify required XHTML Modules the syntax may identify required elements and attributes supported outside of anXHTML Module

the syntax may identify XHTML module preferences



Modularization of XHTML5.3. Syntax for Representing XHTML Modules the syntax may identify attribute value preferences This is not a complete list of functional requirements for a content negotiation syntax -- additionalfunctionality is required to handle declaration of variant resources (related content with different capabilities or features) and user preferences -- but these are beyond the scope of the HTMLWorking Group.

6. Developing DTDs with defined and extended modules This section is informative. The primary purpose of defining XHTML modules and a general modularization methodology isto ease the development of DTDs that are based upon XHTML. These DTDs may extend XHTML by integrating additional capabilities (e.g. [SMIL] [p.60] or [MathML] [p.59] ), or they maydefine a subset of XHTML for use in a specialized device. Regardless of the application, XHTML modules are up to the task. This section describes the techniques that DTD designers must usein order to take advantage of this modularization architecture. It does this by applying the techniques defined in the previous sections in progressively more complex ways, culminating inthe creation of a complete DTD from disparate modules.

Note that in no case do these examples require the modification of the XHTML-provided module files themselves. The XHTML module files are completely parameterized, so that it is possible through separate module definitions and driver files to customize the definition and the contentmodel of each element and each element's hierarchy.

Finally, remember that most users of XHTML are not expected to be DTD authors. DTD authorsare generally people who are defining specialized markup that will improve the readability, simplify the rendering of a document, or ease machine-processing of documents, or they areclient designers that need to define the specialized DTD for their specific client. Consider these cases:

An organization is providing subscriber's information via a web **interface**. The organizationstores its subscriber information in an XML-based database. One way to report that information out from the database to the web is to embed the XML records from thedatabase directly in the XHTML document. While it is possible to merely embed the records, the organization could define a DTD module that describes the records, attach that moduleto an XHTML DTD, and thereby create a complete DTD for the pages. The organization can then access the data within the new elements via the Document Object **Model** [DOM],validate the documents, provide style definitions for the elements that cascade using Cascading Style Sheets [CSS] [p.??] , etc. By taking the time to define the structure of theirdata and create a DTD using the processes defined in this section, the organization can realize the full benefits of XML. An Internet client developer is designing a specialized device. That device will only supporta subset of XHTML, and the devices will always access the Internet via a proxy server that

is validating content before passing it on to the client (to minimize error handling on the



6. Developing DTDs with defined and extended modulesModularization of XHTML client). In order to ensure that the content is valid, the developer creats a DTD that is asubset of XHTML using the processes defined in this section. They then use the new DTD in their proxy server and in their devices, and also make the DTD available to contentdevelopers so that developers can validate their content before making it available. By performing a few simple steps, the client developer can use the modularization architecturedefined in this document to greatly ease their DTD development cost and ensure that they are fully supporting the subset of XHTML that they choose to include.

6.1. Defining additional attributes In some cases, an extension to XHTML can be as simple as additional attributes. Attributes canbe added to an element just by specifying an additional ATTLIST for the element, for example:

would add the "myattr" attribute, with a value type of CDATA, to the "a" element. This worksbecause XML permits the extension of the attribute list for an element at any point in a DTD.

Naturally, adding an attribute to a DTD does not mean that any new **behavior** is defined forarbitrary clients. However, a content developer could use an extra attribute to store information that is accessed by associated scripts via the Document Object **Model** (for example).

6.2. Defining additional elements Defining additional elements is only slightly more complicated than defining additional attributes.Basically, DTD authors should write the element declaration for each element:

After the elements are defined, they need to be integrated into the content **model**. Strategies forintegrating new elements or sets of elements into the content **model** are addressed in Defining the content **model** for a collection of modules [p.45] below.

6.3. Defining a new module When work on extending XHTML modules is done with the intent of making the work generallyavailable for use in developing additional extended DTDs, developers should adhere to the module definition techniques defined in Implementing Document **Model** Modules in the DTD. [p.19] A module constructed using those techniques has several characteristics:



Modularization of XHTML6.1. Defining additional attributes The definitions in the module are related by some common theme that makes it reasonableto have them in a single module. The module is declared such that its entities are uniquely named. The module adheres to a strict naming convention for various classes of definitions thatmake those names predictable.

The module may rely upon entities defined in other modules to specify its entities, elements,and attribute lists. Unless the content **model** of elements in the module is fixed, each element's content modelis parameterized so that it can be extended by DTD authors.

6.4. Defining the content **model** for a collection of modules Since the content **model** of modules is fully parameterized, DTD authors may modify the contentmodel for every element in every module. There are two ways to approach this modification:

1. Re-define the ".content" entity for each element. 2. Re-define one or more of the global content **model** entities (*.class or *.mix).

The strategy taken will depend upon the nature of the modules being combined and the natureof the modules being integrated. The remainder of this section describes techniques for integrating two different classes of modules.

6.4.1. Integrating a stand-alone module into XHTML When a module (and remember, a module can be a collection of other modules) containselements that only reference each other in their content **model**, it is said to be "internally complete". As such, the module can be used on its own (for example, you could define a DTDthat was just that module, and use one of its elements as the root element). Integrating such a module into XHTML is a three step process:

1. Decide what element(s) can be thought of as the root(s) of the new module. 2. Decide where these elements need to attach in the XHTML content tree. 3. Then, for each attachment point in the content tree, add the root element(s) to the contentdefinition for the XHTML elements.

Consider attaching the elements defined above [p.44] . In that example, the element myelement is the root. To attach this element under the object element, and only the objectelement, of XHTML, the following would work:

A DTD defined with this content **model** would allow a document like the following fragment:



6.4. Defining the content **model** for a collection of modulesModularization of XHTML 6.4.2. Mixing a new module throughout the modules in XHTML Extending the example above, to attach this module everywhere that the %Flow.mix contentmodel group is permitted, would require something like the following:

Since the %Misc.class content **model** class is used throughout the XHTML Transitional DTD, thenew module would become available throughout an extended XHTML document.

6.5. Creating a new DTD So far the examples in this section have described the methods of extending XHTML andXHTML's content **model**. Once this is done, the next step is to collect the modules that comprise the DTD into a single DTD driver, incorporating the new definitions so that they override andaugment the basic XHTML definitions as appropriate.

When defining a new DTD, it is essential that each DTD have a unique identifier to use in thexmlns attribute of the root element (usually the

html element). This identifier is often a URI, butin any event is something that can be used by browsers to differentiate the DTD from others.

This identifier is defined using the XHTML1.ns parameter entity when creating a DTD that usesthe XHTML1 structure module.

6.5.1. Creating a simple DTD Using the trivial example above, it is possible to define a new DTD that extends the XHTMLTransitional DTD pretty easily. The following is a complete, working extended DTD:

%XHTML1-t.dtd;



Modularization of XHTML6.5. Creating a new DTD 6.5.2. Creating a DTD by extending XHTML Next, there is the situation where a complete, additional, and complex module is added toXHTML (or to a subset of XHTML). In essence, this is the same as in the trivial example above, the only difference being that the module being added is incorporated in the DTD by referencerather than explicitly including the new definitions in the DTD.

One such complex module is the DTD for [MathML] [p.59] . In order to combine MathML andXHTML into a single DTD, an author would just decide where MathML content should be legal in the document, and add the MathML root element to the content **model** at that point:

%XHTML1-math;

%XHTML1-strict;

Note that, while this is a valid example, it does not create a working DTD at this time. Thereason for this is that the MathML DTD defines two elements (

var and select) that conflictdirectly with XHTML. This conflict needs to be resolved in order for the new DTD to work

correctly.

6.5.3. Creating a DTD by removing and replacing XHTML modules Finally, another way in which DTD authors may use XHTML modules is to define a DTD that is asubset of XHTML (because, for example, they are building devices or software that only supports a subset of XHTML). Doing this is only slightly more complex than the previousexample. The basic steps to follow are:

1. Select the predefined XHTML DTD to use as a basis for the new DTD (Strict, Transitional,or Frameset). 2. Select the modules to remove from that DTD 3. Define a new DTD that "IGNORES" the modules.

For example, consider a device that supports the Strict XHTML 1.0,

but without forms or tables.The DTD for such a device would look like this:



6.5.2. Creating a DTD by extending XHTMLModularization of XHTML

%XHTML1-strict;

Note that this does not actually modify the content **model** for the Strict XHTML 1.0 DTD.However, since XML ignores elements in content models that are not defined, the form and table elements are dropped from the **model** automatically.

6.6. Using the new DTD Once a new DTD has been developed, it can be used in any document. Using the DTD is assimple as just referencing it in the DOCTYPE declaration of a document:

This is an example document using the new elements: A test element

7. Extending XHTML with Compound Documents This section is informative. The use of hybrid DTDs to perform structural validation of extended XHTML documents providesa useful way for documents authors and validating web clients to create XML web pages with capabilities not included in the XHTML 1.0 DTDs. However, in the absence of the functionalityneeded for processing such documents (such as in the case of a non-validating client), how can XHTML be extended? The solution requires that we accept a document authoring mechanismthat is non-validating (not necessarily invalid!) but still provides some level of document integrity and sufficient associated information to assure correct rendering of the document.

7.1. What is a compound document? In the past, HTML was an SGML application, and our discussions regarding its extension wereessentially limited to the addition or modification of new elements or attributes to the HTML DTD, or to the deprecation and removal of existing ones. This in many cases occurred in a proprietary



Modularization of XHTML7. Extending XHTML with Compound Documents way, and often in response to a perceived need in the HTML authoring community for additionalpage functionality or display capabilities, or to accommodate new web technologies. XML however provides us with a different conceptual scheme for dealing with the rapid pace oftechnological change on the web. As the acronym implies, XML has built into it the idea of extensibility and adaptation to a rapidly changing document authoring environment, and soallows for new

technologies and new perceived needs to be accommodated in a standardized way. Previous chapters of this document described a means of extending XHTML that dependson managing changes to multiple DTDs specific to certain document types, very similar to the SGML-style solutions used in the past to extend HTML. XML also allows for the use ofcompound documents, which do not require DTDs or their modification. Compound documents may be defined as documents that contain elements or attributes from multiple document types.

Figure 7.1 A compound document that uses XHTML as its parent document type. In XML, document types are defined in terms of XML Namespaces [XMLNAMES] [p.61] . Thesewill be described in more detail later, but for now we merely note that our definition above for compound documents can be restated in this way:In XML, compound documents are documentsusing more than one XML Namespace.

7.2. The need for compound documents At this point in the development of XML, no sufficiently powerful method has been developed toconfirm the structural validity of compound documents. While several steps are being taken in this direction by W3C, a working solution is still in the beginning stages. Because one of therequirements of such a solution is that it must work in the same fashion for all XML documents,



7.2. The need for compound documentsModularization of XHTML the HTML WG alone cannot take on the responsibility for its development. In the interim, theadmittedly less-than-perfect but still quite useful solution is to create compound documents using XML Namespaces that are only well formed, but whose structural validity cannot bedetermined using current means. While these documents cannot be said to strictly conform to the XHTML specification in the absence of any means of testing their validity, document authorswill benefit greatly from their use.

7.3. Why are compound documents important? Extending XHTML with compound documents provides a solution to the web'sextensibility issues that can be used in a non-validating context. This capability is valuable to document authors who want to create pages that make use of several XML -based markuplanguages in a single document. It is also useful for authors whose intended clients are non-validating, or those who want to create their own elements or component grammars.Another justification for the use of compound documents is in the case of a new and untried technology. The W3C clearly cannot develop a matching specification for each newtechnological advance on the web, nor would it want to do so. Many of these technologies are untested and some will prove not be useful. In order to determine their usefulness however, amethod is needed to be able to easily add the new elements or new grammar to web pages in a standardized way.

Of course the associated semantic functionality of the new elements mustalso be added to the client; how this is done is beyond the scope of this document.

In short, compound documents allow us to quickly and easily extend XML web pages in astandardized way, without any need to resort to modification of DTDs. There are several motivating factors for using compound documents. Among them areinteroperability, scalability, and profiling for both client capabilities and document types. All of these concerns are necessary for effective management of the document life cycle and better workflow:

Interoperability - two aspects of interoperability on the web apply to compound documents.These are device interoperability and document interoperability. At the device level, compound documents allow us to modularize our documents so as to tailor them to specificdevices at **runtime**, without the necessity for creating, modifying or normalizing DTDs. At the document level, we can use compound documents to combine different features of XML inan extensible and standardized way. Scalability - the web has grown and changed at a fantastic rate over the last 5 years.Developing strategies to manage technological change effectively requires that systems be scalable i.e. they must be able to accommodate large changes in the size of the systemover time. XHTML is intended to have the built-in ability to grow with the Net, providing sufficient flexibility and extensibility for the standard to remain stable for a reasonable periodof time. Compound documents allow us to do this effectively in a relatively low-cost way. Profiling - the rate of proliferation of devices designed to access the World-Wide Web isincreasing at a rate far faster than any standards process can hope to track. Content authors cannot hope to create XML web pages for every device with differing capabilities.Tailoring web content for diverse devices requires a modular approach to document



Modularization of XHTML7.3. Why are compound documents important? creation that provides maximum flexibility and automation. While compound DTDs provide apartial solution to this, many situations will require less complexity in processing. Compound documents are both simpler and more flexible.

7.4. Document life-cycle management HTML was designed as a page description language for web pages at a time when the conceptof the WWW was rather different than it is now. At the time of its design, developing methods of document management on a large scale, or for mission-critical purposes, was not a goal.Round-trip document management concepts such as revision control, editing, archiving, and media transformation were not part of HTML's design. The astounding growth of the web, bothquantitatively in terms of number of users and

qualitatively in terms of utility, has created a need for these kinds of extensions to HTML. XML was designed for just this purpose; to provide anextensible way to provide services that HTML did not envision when it was created. HTML is not a document format suitable for storing, authoring or editing documents; its value lies in its abilityto define the structure of web pages as they appear on a user's browser. XML will allow us to create 'the right tool for the right job' for these tasks. Advances such as WebDAV [RFC2518] [p.59] , an XML based markup language designed for distributed web page authoring and versioning, is one such example. Compound documents allow us to provide these services,using XML, in a cost-effective and flexible way. By using appropriate tools for authoring, storage, and revision of our documents (as well as document display) we can fulfill XML's promise for theweb community, extending the lifetime and value of our XML web pages.

7.5. Expected benefits of compound documents At this point it is reasonable to ask why the HTML WG needs two different methodologies forextending XHTML. The method described earlier in this document, using hybrid DTDs, is useful in many cases, and is compatible with DTD based structural validation. Why then the need forcompound documents? The answer is that both compound documents and hybrid DTDs serve a purpose within the HTML authoring community. Design and creation of hybrid DTDs is likely tobe limited to a small and finite group of DTDs created and supported by large vendors and organizations. This provides structural validity at the price of limited extensibility. Implementationmay also require additional expense for publishers, further raising the bar to entry for authors of web pages. It is also less flexible and adaptable to the swift pace of technological change. The compound document approach allows for a more flexible, cheaper means of extendingHTML, while sacrificing the rigorous error checking of structural validity. New XML document types and extensions can be used by authors without inordinate changes to existing standardsand DTDs. Being less formally constrained, it will be easier to incorporate technological advances in our web pages. Authors will not require expensive new tools and training to extendHTML. And since it takes advantage of the natural strengths of XML for extensibility, it is a standardized solution, giving page authors the much needed stability needed for long term webpublishing. Compound documents are intended to be an interim solution, useful for some classes of documents and authors, less rigid than formally defined document types using DTDsbut more structured and easily automated than today's 'broken' HTML pages.



7.4. Document life-cycle managementModularization of XHTML Overall, the benefits of using compound documents overlap those of hybrid documents to alarge degree:

A wider audience More reliable presentation Better authoring and workflow Standardized method of extending HTML simply and easily Easily supported by today's software More flexible and cheaper than hybrid DTDs

7.6. Creating and Using Compound Documents Extending HTML with compound documents requires an understanding of the XML 1.0specification and XML Namespaces. While authoring compound documents will require more skill and work than documents without extensions, this small amount of additional complexity willrepay page authors handsomely in terms of useful extended documents.

7.6.1. The XML family tree XML, in its simplest definition, is a set of rules for creating new document types. Using XML. sstrict formalism, we can construct markup languages designed for a specific purpose, such as creating vector graphics, tracking revisions, and formatting mathematical equations. XML hasbeen conceived by the W3C as a reference platform for the creation of markup languages for specific web needs. These markup languages are often called . XML dialects. or . XMLcomponent grammars. . Figure 2 shows the XML family tree, including many XML component grammars that either are currently in development or are already standardized by W3C.Extensibility in XML terms implies that these grammars can be used in XML web pages in a modular fashion, similar to the way software components can be added (or removed) frommodern software packages.




Figure 7.2 The XML Family Tree 7.6.2. XHTML as a parent document type Using XML component grammars together in the same web page is what we mean when werefer to "compound documents". These documents clearly defy definition as being one document type or another in the way document types are defined in SGML. XHTML, as thesuccessor markup language to HTML, is expected to be the document type that the majority of users want to extend. It seems quite likely that many more page authors will want to addfeatures from other component grammars to XHTML, rather than vice-versa. This means that XHTML is likely to be the most commonly extended parent document type. The parentdocument type of a compound document is the document type of the document's root element.

It is, however, perfectly reasonable for authors to want to reverse this process and add XHTMLcomponents to their XML documents of other types. XHTML tables or forms might be a good example of a domain-specific set of elements whose functionality can be reused rather thanreinvented. The modularization of XHTML should provide for the possibility of using XHTML in compound documents whose parent document is not XHTML. We can also see that component grammars are not necessarily required to be complete markuplanguages, but may be a

modularized subset of an existing XML based markup language

7.6.3. Using multiple namespaces The XML Namespaces specification is separate from the XML 1.0 specification, having beenreleased after the completion of XML 1.0. The Namespaces specification provides a preliminary method of disambiguating element (and attribute) names in XML documents.



7.6.2. XHTML as a parent document typeModularization of XHTML 7.6.3.1. Summary of XML Namespaces The purpose of XML Namespaces is to provide an adequate scoping mechanism for elements incompound documents. The XML Namespaces methodology requires that a new attribute "xmlns" be added to define a namespace identifier for all document elements. In the case ofcompound documents, multiple namespace attributes are required. These namespace attributes define a prefix for element and attribute names from that namespace, thereby distinguishingthem from elements from other document types. In general, each XML Namespace is specific to a particular document type. 7.6.3.2. Using namespace prefixes Namespaces can be assigned to any element at any point in the document, but in order to aiddocument parsing are generally declared on the document's root element. Elements are then prefixed (or 'colonized') with the defined prefix to indicate their parent namespace. A documentmay have a default namespace, which does not require a prefix.

This example displays the text "Hello World" and then a rectangle 100 pixels on each side,located at 100,100 and colored red.

...

Hello World!

This document defines xhtml1-strict as its default namespace; elements from this namespaceare not prefixed. It also declares an additional namespace for some SVG (scalable vector graphics) [REC-SVG] [p.59] elements. These elements are prefixed using the XML Namespacesnotation. The 'SVG' prefix indicates the namespace in use by the rect element. All elements and attributes from other namespaces must be prefixed to prevent ambiguity. Note that the value of the xmlns attribute is a URI [RFC2396] [p.60] . The XML Namespacesspecification says that this value must be a valid URI. However, it is not necessarily true that the URI used will actually serve as a locator of an associated resource (like a DTD or profile). Thenamespace URI, according to the specification, serves only as a unique identifier for that namespace. Accordingly, software may use the namespace URI to disambiguate element andattribute names, but may not depend on traversing the URI in order to obtain further

information. Further work on XML Namespaces may make use of the URI properties of namespaces as partof some future validation scheme.

7.6.4. Validity v. well-formedness HTML, as an SGML application, inherited a method of defining and checking structural integritybased on the notion of document types. Documents that identified themselves as being of a specific document type could be checked for errors or 'validity' against a DTD or Document TypeDefinition. Each distinct document type in SGML has its own DTD. Because compound



 documents are by definition composed of more than one document type, DTDs cannot beutilized to determine their validity. Document types cannot be mixed in SGML, so compound documents cannot be validated. (SGML provides for a 'subdoc' facility similar to compounddocuments as discussed here that is not in wide use.)

7.6.4.1. Limitations on the usefulness of validity for compound documents While traditional DTD validation cannot be used with compound documents, issues of structuralintegrity are still of importance, especially to authors seeking some measure of insurance that their documents will be displayed properly by user agents. XML provides an incrementalapproach to checking structural integrity, using the concepts of well-formedness and validity to define different stages of document checking, depending on the nature of the needs of theauthor. Compound documents as defined here are not valid XHTML documents, but are well-formed. This means that at a minimum, elements in compound documents must be nestedcorrectly; conform to proper XML character usage, and properly use attributes. It is worth repeating that compound documents cannot be valid XHTML documents. Authors wanting toextend XHTML at this point in time must chose to either modify the XHTML DTDs as described earlier in this document, or make use of compound documents and XML Namespaces, therebygiving up the ability to determine structural validity using DTDs.

7.6.5. Fragment validation A compound document can be thought of as consisting of a parent document in which one ormore XML document fragments are embedded (fig. 7.1). Considered in this fashion, each individual fragment out of which a document is composed might be validated separately, andthen the whole composed into the final document. Thus the author could be assured that the different parts of the document each had their own individual structural integrity. This is acomplex subject in itself and is the proper responsibility of the XML Fragment Interchange Working Groups [FRAG] [p.??] and is under discussion in other document areas as well [TC9601] [p.62] .

Validation using DTDs does not require that each document validated begin with the document'sroot element. In fact, DTDs do not specify

the root elements of documents explicitly; the document's DOCTYPE identifier is intended to be used for this purpose. Fragment validation asdescribed here is essentially conceptual; this is a rather brief and simplistic overview.

7.6.6. Nesting depth of namespaces The embedding of document fragments of one type into documents of other types leads to the'nesting problem' for XML documents. It appears clear that without imposed constraints, it is possible for a document of type a to contain a document fragment of type b, which contains adocument fragment of type c, which contains...ad infinitum. This presents some difficulty for parsers attempting to process documents as a stream, which is required by XML 1.0 . While noarbitrary nesting depth can be imposed externally, authors are advised that many levels of nesting may increase processing times for their documents.

- 55 -

7.6.5. Fragment validationModularization of XHTML 7.7. Current support status Software that complies with the XML 1.0 specification and the XML Namespaces specificationwill support compound documents as described here. Additionally the software must support the particular semantics of the elements (and their attributes) used in a particular compounddocument. Clearly every software application will not support every possible XML component grammar. In effect, software will provide support for specific XML Namespaces, which can thenbe used in compound documents reliably. In the case of a document containing elements contained in namespaces not supported by a given piece of software, the software should, in thetime-honored fashion of the web, simply ignore the elements that it does not understand.

While software response to elements from an unsupported namespace is certainlyimplementation specific, a best effort attempt should be made to render the page, so long as it is a well formed XML document. This line of thought will lead you to the conclusion that there are two kinds of XML elements;those with purely presentational qualities and those with deeper semantic qualities that cannot be specified in a stylesheet. Examples might be the or  elements, which havesemantics whose result for the user is outside the scope a stylesheet (or a DTD). Purely structural/presentational elements are simple to add to a DTD in this fashion; trivial really. butthey are essentially just containers for hanging a CSS style rule on. Why have more than one? Deeper semantic modules (such as SVG, or forms) have meaning in the document context thatrequires the browser to have accompanying functionality. Without this functionality, the document may well validate but won't work; be purely conforming and result in an error.

7.7.1. Browser support One major browser manufacturer [MSIE5] [p.62]

currently supports compound documents asdescribed here, essentially in full. Many component grammars are currently under development, in different stages of development, making support for them difficult if not impossible. Howeverin the future it is expected that all popular browser makers will support compound documents in some fashion.

7.7.2. Editor/Tool support XML tools and editors are currently in the developmental stage. The XML Namespacesspecification was only recently approved as a specification, so software vendors are currently working to fully support it. Some of the facilities described here, such as validation of documentfragments, are available in existing software. Hopefully XML' s ease of automation and widespread use will encourage software makers to provide further support for XML in their products.



Modularization of XHTML7.7. Current support status 7.7.3. Simple examples The compound document examples shown here all require the use of supporting software to beviewed properly. With a very limited set of component grammars fully completed, support for them is limited to specific preliminary implementations. Because this draft is a work in progress,and the status of other XML component grammars, these examples will become obsolete quickly, and so should be considered as examples intended to illustrate the concepts of usingcompound documents rather than examples that are intended to work 'out of the box.'.

(Note that because compound XHTML documents (compound documents with an XHTML rootelement) are not intended to be 'valid' in the SGML sense of structural validity, it is not useful to include a standard DOCTYPE identifier for the document.) 7.7.3.1. An example using XHTML and SVG (SVG stands for Scalable Vector Graphics, an XML component grammar that is still underdevelopment by W3C.)

This example would display a pie chart showing quarterly sales by region, along with someappropriate text.

...

The following pie chart displays this quarter's sales in different regions. As you can see, the company is doing well.

In this example, I've explicitly prefixed all the document elements to make clear the compoundnature of the document. This web page would display some XHTML text along with a pie chart showing a SVG vector graphic with labels and colored pie sections. (Thanks to the SVGWorking Group, from whom this SVG code was borrowed.)

7.7.3. Simple examplesModularization of XHTML 7.7.4. Using XHTML as an XML component grammar XHTML, being just one of many XML component grammars, may be used by authors incompound documents of a different parent type. This is anticipated in the case where the author wants to add structured text to their documents, or XHTML tables or forms, especially fortransmission over the Web. This works no differently than in any other compound document. The modular, component-based design of the XML family of languages provides the ability to'mix and match' elements from a large variety of XML document types in order to create fully functional web pages. 7.7.4.1. Using simple presentation elements from XHTML An example using XHTML elements in a MathML document TBD

7.8. Future Work Several W3C Working Groups are tasked with developing parts of a more comprehensivestructural validation mechanism for XML. Among these are the XML Syntax Group, under whose purview falls XML Namespaces; the XML Fragment Interchange Working Group [REC-FRAG] [p.62] , which deals with processing of XML fragments outside their document context, and the XML Schema Working Group [XSCHEMA] [p.62] , now considering several proposals for anXML component grammars to replace DTDs as a means of determining structural validity.

7.9. Acknowledgements The author would like to thank Casey Caston (caseyc@cnet.com) for creating the images usedin this chapter.

A. References This appendix is normative.

A.1. Normative References [XHTML1] Extensible HTML (XHTML) 1.0: W3C Working Draft, Steven Pemberton, et. al., 4 March

1999.See: http://www.w3.org/TR/WD-html-in-xml [XML] Extensible Markup Language (XML) 1.0: W3C Recommendation, Tim Bray, Jean Paoli, C.

M. Sperberg-McQueen, 10 February 1998.See: http://www.w3.org/TR/REC-xml



Modularization of XHTMLA. References [HTML40] HTML 4.0 Specification: W3C Recommendation, Dave Raggett, Arnaud Le Hors, Ian

Jacobs, 24 April 1998.See: http://www.w3.org/TR/REC-html40 [SGML] Information Processing -- Text and Office Systems -- Standard Generalized Markup

Language (SGML), ISO 8879:1986.Please consult

http://www.iso.ch/cate/d16387.html for information about the standard, or http://www.oasis-open.org/cover/general.html#overview about SGML. [MATHML] Mathematical Markup Language (MathML) 1.0 Specification: W3C Recommendation,Stephen Buswell, Stan Devitt et al, 7 April 1998. See: http://www.w3.org/TR/REC-MathML [WEBDAV] HTTP Extensions for Distributed Authoring -- WEBDAV Y. Goland, et. al., February 1999See: http://www.ietf.org/rfc/rfc2518.txt [REC SVG] Scalable Vector Graphics (SVG) Specification - Working Draft Jon Ferraiolo, et. al., 11

February, 1999 See: http://www.w3.org/TR/WD-SVG/

A.2. Informative References [CATALOG] Entity Management: OASIS Technical Resolution 9401:1997 (Amendment 2 to TR 9401)

Paul Grosso, Chair, Entity Management Subcommittee, SGML Open, 10 September 1997.See: http://www.oasis-open.org/html/a401.htm [DEVDTD] Developing SGML DTDs: From Text to **Model** to Markup, Eve Maler and Jeanne El

Andaloussi.Prentice Hall PTR, 1996, ISBN 0-13-309881-8. [STRUCTXML] Structuring XML Documents, David Megginson. Part of the Charles Goldfarb Series on

Information Management.Prentice Hall PTR, 1998, ISBN 0-13-642299-3. [SGML-XML] Comparison of SGML and XML: W3C Note, James Clark, 15 December 1997.

See: http://www.w3.org/TR/NOTE-sgml-xml-971215 [XLINK] XML Linking Language (XLink): W3C Working Draft, Eve Maler and Steve DeRose, 3March 1998. A new XLink requirements document is expected soon, followed by a working draft update. See: http://www.w3.org/TR/WD-xlink [DOCBOOK] DocBook DTD, Eve Maler and Terry Allen.

Originally created under the auspices of the Davenport Group, DocBook is now maintained



A.2. Informative ReferencesModularization of XHTML by OASIS. The Customizer's Guide for the DocBook DTD V2.4.1 is available from this site.See: http://www.oasis-open.org/docbook/index.html [DUBLIN] The Dublin Core: A Simple Content Description **Model** for Electronic Resources, The Dublin

Core Metadata Initiative.See: http://purl.oclc.org/dc/ [HTML32] HTML 3.2 Reference Specification: W3C Recommendation, Dave Raggett, 14 January

1997.See: http://www.w3.org/TR/REC-html32 [ISO-HTML] ISO/IEC 15445:1998 HyperText Markup Language (HTML), David M. Abrahamson and

Roger Price.See: http://dms1.cs.uml.edu/15445/FinalCD.html [RDF] Resource Description Framework (RDF): **Model** and Syntax Specification, Ora Lassila and

Ralph R. Swick, 19 August 1998.See: http://www.w3.org/TR/PR-rdf-syntax [SMIL] Synchronized Multimedia Integration Language (SMIL) 1.0 Specification, Philipp Hoschka,

15 June 1998.See: http://www.w3.org/TR/REC-smil [TEI] The Text Encoding Initiative (TEI)

See: http://www.uic.edu/orgs/tei/ [URI] Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L.Masinter, August 1998. See: http://www.ietf.org/rfc/rfc2396.txt. This RFC updates RFC >1738 [URL] [p.??] and [RFC1808] [p.??] . [URL] IETF RFC 1738, Uniform Resource Locators (URL), T. Berners-Lee, L. Masinter, M.

McCahill.See: http://www.ietf.org/rfc/rfc1738.txt [RFC-1808] Relative Uniform Resource Locators, R. Fielding.

See: http://www.ietf.org/rfc/rfc1808.txt [CC/PP] "Composite Capability/Preference Profiles (CC/PP): A user side framework for contentnegotiation", F. Reynolds, J. Hjelm, S. Dawkins, S. Singhal, 30 November 1998. This document describes a method for using the Resource Description Format (RDF) tocreate a general, yet extensible framework for describing user preferences and device capabilities. Servers can exploit this to customize the **service** or content provided.Available at: http://www.w3.org/TR/NOTE-CCPP [CSS2] "Cascading Style Sheets, level 2 (CSS2) Specification", B. Bos, H. W. Lie, C. Lilley, I.



Modularization of XHTMLA.2. Informative References Jacobs, 12 May 1998.Available at: http://www.w3.org/TR/REC-CSS2 [DOM] "Document Object **Model** (DOM) Level 1 Specification", Lauren Wood et al., 1 October

1998.Available at: http://www.w3.org/TR/REC-DOM-Level-1 [ERRATA] "HTML 4.0 Specification Errata".

This document lists the errata for the HTML 4.0 specification.Available at: http://www.w3.org/MarkUp/html40-updates/REC-html40-19980424-errata.html [HTML] "HTML 4.0 Specification", D. Raggett, A. Le Hors, I. Jacobs, 18 December 1997,

revised 24

April 1998.Available at: http://www.w3.org/TR/REC-html40 [POSIX.1] "ISO/IEC 9945-1:1990 Information Technology - Portable Operating System **Interface**

(POSIX) - Part 1: System Application Program **Interface** (API) [C Language]", Institute ofElectrical and Electronics Engineers, Inc, 1990. [RFC2119] "RFC2119: Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March

1997.Available at: http://www.ietf.org/rfc/rfc2119.txt [RFC2376] "RFC2376: XML Media Types", E. Whitehead, M. Murata, July 1998.

Available at: http://www.ietf.org/rfc/rfc2376.txt [RFC2396] "RFC2396: Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, L.Masinter, August 1998. This document updates RFC1738 and RFC1808.Available at: http://www.ietf.org/rfc/rfc2396.txt [TIDY] "HTML Tidy" is a tool for detecting and correcting a wide range of markup errors prevalent

in HTML. It can also be used as a tool for converting existing HTML content to be wellformed XML. Tidy is being made available on the same terms as other W3C sample code, i.e. free for any purpose, and entirely at your own risk.It is available from: http://www.w3.org/Status.html#TIDY [XMLNAMES] "Namespaces in XML", T. Bray, D. Hollander, A. Layman, 14 January 1999.

XML namespaces provide a simple method for qualifying names used in XML documentsby associating them with namespaces identified by URI. Available at: http://www.w3.org/TR/REC-xml-names [XMLSTYLE] "Associating stylesheets with XML documents Version 1.0", J. Clark, 14 January 1999.This document describes a means for a stylesheet to be associated with an XML document by including one or more processing instructions with a target of xml-stylesheet in thedocument's prolog.



A.2. Informative ReferencesModularization of XHTML Available at: http://www.w3.org/TR/PR-xml-stylesheet [FRMWRK] "Protocol-independent content negotiation framework", Klyne G., 16 February 1999.See: http://www.ietf.org/internet-drafts/draft-ietf-conneg-requirements-02.txt [SYNTAX] "A syntax for describing media feature sets", Klyne G., 14 December 1998.

See http://www.ietf.org/internet-drafts/draft-ietf-conneg-feature-syntax-04.txt [URLSYN] "Syntax extensions for abbreviating media feature sets with URLs", Newman W., 26February 1999. See

http://www.ietf.org/internet-drafts/draft-ietf-conneg-feature-sets-at-urls-00.txt [XMLMOD] "XML Modularization of HTML 4.0", M. Altheim, Sun Microsystems, 2 February 1999See http://www.altheim.com/specs/xhtml/NOTE-xhtml-modular..html [REC-FRAG] XML Fragment Interchange - Working Draft Paul Grosso, et. al., 3 March, 1999

See: http://www.w3.org/TR/WD-xml-fragment [TC9601] SGML standard Technical Corrigendum 9601 Paul Grosso??See: [MSIE5] Microsoft Internet Explorer Version 5.0

See: http://www.microsoft.com/windows/ie/ie5/default.asp [XSCHEMA] XML Schema Requirements - W3C Note Ashok Malhotra, et. al., 15 February 1999See: http://www.w3.org/TR/1999/NOTE-xml-schema-req-19990215

B. Design Goals This appendix is informative There are six major design goals for the modularization framework for XHTML:

[G1] Provide a means for the W3C and third parties to integrate XHTML into other XMLlanguages.

[G2] Provide a means for the W3C to extend XHTML with new or optional features. [G3] Provide a means for third parties to extend XHTML with domain-specific features. [G4] Provide a means for third parties to integrate other XML languages into XHTML. [G5] Improve the ability to create a close approximation to the HTML 4.0 DTDs. [G6] Improve ease-of-use for DTD developers.



Modularization of XHTMLB. Design Goals B.1. Requirements The design goals listed in the previous section lead to a large number of requirements for themodularization framework. These requirements, summarized in this section, can be further classified according to the major features of the framework to be described.

B.1.1. Granularity Collectively the requirements in this section express the desire that the modules defined withinthe framework hit the right level of granularity:

[R1.1] Semantic modules should promote and maintain content portability. [R1.2] Semantic modules should promote platform profile standardization. [R1.3] Semantic modules should be large enough to promote interoperability. [R1.4] Semantic modules should be small enough to avoid the need for subsets. [R1.5] Semantic modules should collect elements with similar or related semantics. [R1.6] Semantic modules should separate elements with dissimilar or unrelated semantics. [R1.7] Modules should be small enough to allow single

element document type modules.

B.1.2. Composibility The composibility requirements listed here are intended to ensure that the modularizationframework be able to express the right set of target modules required by the communities that will be served by the framework:

[R2.1] The module framework should allow construction of semantic modules for XHTML1.0.

[R2.2] The module framework should allow construction of semantic modules that closelyapproximate HTML 4.0. [R2.3] The module framework should allow construction of semantic modules for other W3CRecommendations. [R2.4] The module framework should allow construction of semantic modules for other XMLdocument types. [R2.5] The module framework should allow construction of semantic modules for a widerange of platform profiles.

B.1.3. Ease of Use The modularization framework will only receive widespread adoption if it describes mechanismsthat make it easy for our target audience to use the framework:

[R3.1] The module framework should make it easy for document type designers to subsetand extend XHTML semantic modules. [R3.2] The module framework should make it easy for document type designers to createsemantic modules for other XML document types.



 [R3.3] The module framework should make it easy for document authors to validateelements from different semantic modules. B.1.4. Compatibility The intent of this document is that the modularization framework described here should workwell with the XML and other standards being developed by the W3C Working Groups:

[R4.1] The module framework should strictly conform to the XML 1.0 Recommendation. [R4.2] The module framework should be compatibile with the XML linking specification. [R4.3] The module framework should be compatibile with the XML stylesheet specification. [R4.4] The module framework should be able to adopt new W3C recommendations whereappropriate.

[R4.5] The module framework should not depend on W3C work in progress. [R4.6] The module framework should not depend on work done outside W3C.

B.1.5. Conformance The effectiveness of the framework will also be measured by how easy it is to test the behaviorof modules developed

according to the framework, and to test the documents that employ those modules for validation:

[R5.1] It should be possible to validate documents constructed using elements andattributes from semantic modules.

[R5.2] It should be possible to explicitly describe the **behavior** of elements and attributesfrom semantic modules. [R5.3] It should be possible to verify the **behavior** of elements and attributes from semanticmodules. [R5.4] It should be possible to verify a compound document type as an XHTML documenttype. [R5.5] Modules defined in accordance with the methods in this document shall not duplicatethe names of elements or parameter entities defined in XHTML modules.

C. XHTML Document Type Definitions C.1. SGML Open Catalog for XHTML This section contains the SGML Open Catalog-format definition of the various FPIs for XHTML.

```
--
.....................................................................
-- -- File
catalog ..............................................................
--
```

-- XHTML 1.0 (HTML 4.0-Based XML Version) Catalog Data File Revision: @(#)XHTML1.cat 1.19 99/04/01 SMI This is the catalog data file for various versions of the XHTML DTD.



Modularization of XHTMLC. XHTML Document Type Definitions You do not need to use the file names listed here, and do not need to use the filename method of identifying storage objects at all.

See "Entity Management", SGML Open Technical Resolution 9401 for detailed information on supplying and using catalog data. This document is available from OASIS at URL:

```
--

--
.....................................................................
-- -- SGML declaration associated with
XHTML ................................. --

OVERRIDE YES SGMLDECL "xml1.dcl" -- for use with non-Unicode
compatible parsers: -- -- SGMLDECL "xml1n.dcl" --

--
```

```
.....................................................................
-- -- XHTML 1.0 DTD driver
files ..................................... --

PUBLIC "-//W3C//DTD XHTML 1.0//EN" "XHTML1-s.dtd" PUBLIC "-//W3C//DTD
XHTML 1.0 Strict//EN" "XHTML1-s.dtd" PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "XHTML1-t.dtd" PUBLIC "-//W3C//DTD XHTML 1.0
Frameset//EN" "XHTML1-f.dtd"

--
.....................................................................
-- -- XHTML 1.0
modules ..................................................... --

PUBLIC "-//W3C//ENTITIES XHTML 1.0 Common Names//EN" "XHTML1-
names.mod" PUBLIC "-//W3C//ENTITIES XHTML 1.0 Character Entities//EN"
"XHTML1-charent.mod" PUBLIC "-//W3C//ENTITIES XHTML 1.0 Intrinsic
Events//EN" "XHTML1-events.mod" PUBLIC "-//W3C//ENTITIES XHTML 1.0
Common Attributes//EN" "XHTML1-attribs.mod"

PUBLIC "-//W3C//ELEMENTS XHTML 1.0 Document Model//EN" "XHTML1-
model.mod" PUBLIC "-//W3C//ELEMENTS XHTML 1.0 Inline Structural//EN"
"XHTML1-inlstruct.mod" PUBLIC "-//W3C//ELEMENTS XHTML 1.0 Inline
Presentational//EN" "XHTML1-inlpres.mod" PUBLIC "-//W3C//ELEMENTS
XHTML 1.0 Inline Phrasal//EN" "XHTML1-inlphras.mod" PUBLIC "-
//W3C//ELEMENTS XHTML 1.0 Block Structural//EN" "XHTML1-
blkstruct.mod" PUBLIC "-//W3C//ELEMENTS XHTML 1.0 Block
Presentational//EN" "XHTML1-blkpres.mod" PUBLIC "-//W3C//ELEMENTS
XHTML 1.0 Block Phrasal//EN" "XHTML1-blkphras.mod"

PUBLIC "-//W3C//ELEMENTS XHTML 1.0 Scripting//EN" "XHTML1-script.mod"
PUBLIC "-//W3C//ELEMENTS XHTML 1.0 Stylesheets//EN" "XHTML1-
style.mod" PUBLIC "-//W3C//ELEMENTS XHTML 1.0 Object Element//EN"
"XHTML1-object.mod" PUBLIC "-//W3C//ELEMENTS XHTML 1.0 Client-side
Image Map//EN" "XHTML1-csismap.mod" PUBLIC "-//W3C//ELEMENTS XHTML
1.0 Linking//EN" "XHTML1-linking.mod" PUBLIC "-//W3C//ELEMENTS XHTML
1.0 Images//EN" "XHTML1-image.mod" PUBLIC "-//W3C//ELEMENTS XHTML 1.0
Lists//EN" "XHTML1-list.mod" PUBLIC "-//W3C//ELEMENTS XHTML 1.0
Forms//EN" "XHTML1-form.mod" PUBLIC "-//W3C//ELEMENTS XHTML 1.0
Tables//EN" "XHTML1-table.mod" PUBLIC "-//W3C//ELEMENTS XHTML 1.0
Metainformation//EN" "XHTML1-meta.mod" PUBLIC "-//W3C//ELEMENTS XHTML
1.0 Document Structure//EN" "XHTML1-struct.mod"

--
.....................................................................
-- -- XHTML Frameset or Transitional
modules ................................... --

-- (not needed for delivery of XHTML 1.0 Strict) -- PUBLIC "-
```

//W3C//ENTITIES XHTML 1.0 Transitional Attributes//EN" "XHTML1-attribs-t.mod"



C.1. SGML Open Catalog for XHTMLModularization of XHTML PUBLIC "-//W3C//ELEMENTS XHTML 1.0 Transitional Document **Model**//EN" "XHTML1-**model**-t.mod" PUBLIC "-//W3C//ELEMENTS XHTML 1.0 Java Applets//EN" "XHTML1-applet.mod" PUBLIC "-//W3C//ELEMENTS XHTML 1.0 Frames//EN" "XHTML1-frames.mod"

```
--
.....................................................................
-- -- XHTML 1.0 entity
sets ..................................................... --

PUBLIC "-//W3C//ENTITIES Latin 1//EN//XML" "XHTML1-lat1.ent" PUBLIC
"-//W3C//ENTITIES Special//EN//XML" "XHTML1-special.ent" PUBLIC "-
//W3C//ENTITIES Symbols//EN//XML" "XHTML1-symbol.ent"

--
.....................................................................
-- -- XHTML 1.0 Architectural Forms
Module ..................................... --

PUBLIC "-//W3C//ELEMENTS XHTML 1.0 Base Architecture//EN" "XHTML1-
arch.mod" --
.....................................................................
-- -- XHTML 1.0 Experimental Extension DTDs and
Modules ........................ --

PUBLIC "-//W3C//DTD XHTML 1.0 Extension - MathML//EN" "XHTML1-m.dtd"
PUBLIC "-//W3C//DTD XHTML 1.0 MathML//EN" "XHTML1-math.mod" PUBLIC "-
//W3C//ELEMENTS XHTML 1.0-Based Subset Simplified Tables//EN"
"XHTML1-table32.mod" PUBLIC "-//W3C//ELEMENTS XHTML 1.0-Based Subset
Simplified Forms//EN" "XHTML1-form32.mod"

-- End of catalog
data ..................................................... -- --
.....................................................................
--
```

C.2. SGML Declaration for XHTML This section contains the SGML Declaration that supports the XHTML DTDs.

```
CHARSET BASESET "ISO Registration Number 176//CHARSET ISO/IEC 10646-
1:1993 UCS-4 with implementation level 3//ESC 2/5 2/15 4/6" DESCSET 0
9 UNUSED 9 2 9 11 2 UNUSED 13 1 13 14 18 UNUSED 32 95 32 127 1 UNUSED
128 32 UNUSED 160 55136 160 55296 2048 UNUSED -- surrogates -- 57344
```

8190 57344 65534 2 UNUSED -- FFFE and FFFF -- 65536 1048576 65536

- 66 -

Modularization of XHTMLC.2. SGML Declaration for XHTML CAPACITY NONE -- Capacities are not restricted in XML -- SCOPE DOCUMENT SYNTAX SHUNCHAR NONE BASESET "ISO Registration Number 176//CHARSET ISO/IEC 10646-1:1993 UCS-4 with implementation level 3//ESC 2/5 2/15 4/6" DESCSET 0 1114112 0 FUNCTION RE 13 RS 10 SPACE 32 TAB SEPCHAR 9 ITAB SEPCHAR 12288 -- ideographic space --

NAMING LCNMSTRT "" UCNMSTRT "" NAMESTRT 58 95 192-214 216-246 248-305 308-318 321-328 330-382 384-451 461-496 500-501 506-535 592-680 699-705 902 904-906 908 910-929 931-974 976-982 986 988 990 992 994-1011 1025-1036 1038-1103 1105-1116 1118-1153 1168-1220 1223-1224 1227-1228 1232-1259 1262-1269 1272-1273 1329-1366 1369 1377-1414 1488-1514 1520-1522 1569-1594 1601-1610 1649-1719 1722-1726 1728-1742 1744-1747 1749 1765-1766 2309-2361 2365 2392-2401 2437-2444 2447-2448 2451-2472 2474-2480 2482 2486-2489 2524-2525 2527-2529 2544-2545 2565-2570 2575-2576 2579-2600 2602-2608 2610-2611 2613-2614 2616-2617 2649-2652 2654 2674-2676 2693-2699 2701 2703-2705 2707-2728 2730-2736 2738-2739 2741-2745 2749 2784 2821-2828 2831-2832 2835-2856 2858-2864 2866-2867 2870-2873 2877 2908-2909 2911-2913 2949-2954 2958-2960 2962-2965 2969-2970 2972 2974-2975 2979-2980 2984-2986 2990-2997 2999-3001 3077-3084 3086-3088 3090-3112 3114-3123 3125-3129 3168-3169 3205-3212 3214-3216 3218-3240 3242-3251 3253-3257 3294 3296-3297 3333-3340 3342-3344 3346-3368 3370-3385 3424-3425 3585-3630 3632 3634-3635 3648-3653 3713-3714 3716 3719-3720 3722 3725 3732-3735 3737-3743 3745-3747 3749 3751 3754-3755 3757-3758 3760 3762-3763 3773 3776-3780 3904-3911 3913-3945 4256-4293 4304-4342 4352 4354-4355 4357-4359 4361 4363-4364 4366-4370 4412 4414 4416 4428 4430 4432 4436-4437 4441 4447-4449 4451 4453 4455 4457 4461-4462 4466-4467 4469 4510 4520 4523 4526-4527 4535-4536 4538 4540-4546 4587 4592 4601 7680-7835 7840-7929 7936-7957

- 67 -

C.2. SGML Declaration for XHTMLModularization of XHTML 7960-7965 7968-8005 8008-8013 8016-8023 8025 8027 8029 8031-8061 8064-8116 8118-8124 8126 8130-8132 8134-8140 8144-8147 8150-8155 8160-8172 8178-8180 8182-8188 8486 8490-8491 8494 8576-8578 12295 12321-12329 12353-12436 12449-12538 12549-12588 19968-40869 44032-55203

LCNMCHAR "" UCNMCHAR "" NAMECHAR 45-46 183 720-721 768-837 864-865 903 1155-1158 1425-1441 1443-1465 1467-1469 1471 1473-1474 1476 1600 1611-1618 1632-1641 1648 1750-1764 1767-1768 1770-1773 1776-1785 2305-2307 2364 2366-2381 2385-2388 2402-2403 2406-2415 2433-2435 2492 2494-2500 2503-2504 2507-2509 2519 2530-2531 2534-2543 2562 2620 2622-2626 2631-2632 2635-2637 2662-2673 2689-2691 2748 2750-2757 2759-2761 2763-2765 2790-2799 2817-2819 2876 2878-2883 2887-2888

2891-2893 2902-2903 2918-2927 2946-2947 3006-3010 3014-3016 3018-3021 3031 3047-3055 3073-3075 3134-3140 3142-3144 3146-3149 3157-3158 3174-3183 3202-3203 3262-3268 3270-3272 3274-3277 3285-3286 3302-3311 3330-3331 3390-3395 3398-3400 3402-3405 3415 3430-3439 3633 3636-3642 3654-3662 3664-3673 3761 3764-3769 3771-3772 3782 3784-3789 3792-3801 3864-3865 3872-3881 3893 3895 3897 3902-3903 3953-3972 3974-3979 3984-3989 3991 3993-4013 4017-4023 4025 8400-8412 8417 12293 12330-12335 12337-12341 12441-12442 12445-12446 12540-12542

NAMECASE GENERAL NO ENTITY NO

DELIM GENERAL SGMLREF HCRO "&#x" -- 38 is the number for ampersand -- NESTC "/" NET ">" PIC "?>" SHORTREF NONE

NAMES SGMLREF

QUANTITY NONE -- Quantities are not restricted in XML --

ENTITIES "amp" 38 "lt" 60 "gt" 62 "quot" 34 "apos" 39

- 68 -

Modularization of XHTMLC.2. SGML Declaration for XHTML FEATURES MINIMIZE DATATAG NO OMITTAG NO RANK NO SHORTTAG -- SHORTTAG is needed for NET -- STARTTAG EMPTY NO UNCLOSED NO NETENABL IMMEDNET ENDTAG EMPTY NO UNCLOSED NO ATTRIB DEFAULT YES OMITNAME NO VALUE NO EMPTYNRM YES IMPLYDEF ATTLIST YES DOCTYPE YES ELEMENT YES ENTITY YES NOTATION YES LINK SIMPLE NO IMPLICIT NO EXPLICIT NO OTHER CONCUR NO SUBDOC NO FORMAL NO URN NO KEEPRSRE YES VALIDITY TYPE ENTITIES REF ANY INTEGRAL YES APPINFO NONE SEEALSO "ISO 8879//NOTATION Application Requirements for XML//EN" >

C.3. XHTML Character Entities XHTML DTDs make available a standard collection of named character entities. Those entitiesare defined in this section.

- 69 -

C.3. XHTML Character EntitiesModularization of XHTML C.3.1. XHTML Latin 1 Character Entities

- 72 -

Modularization of XHTMLC.3.1. XHTML Latin 1 Character Entities

- 73 -

C.3.1. XHTML Latin 1 Character EntitiesModularization of XHTML

C.3.2. XHTML Special Characters

- 74 -

Modularization of XHTMLC.3.2. XHTML Special Characters

C.3.3. XHTML Mathematical, Greek, and Symbolic Characters

- 78 -

Modularization of XHTMLC.3.3. XHTML Mathematical, Greek, and Symbolic Characters

C.4. Common Declaration In order to take advantage of the XHTML DTD Modules, DTD authors needs to define thecontent **model** for their DTD. XHTML provides a variety of tools to ease this effort. They are defined in the following support modules. See Extending XHTML [p.??] for more information onusing these and the Modules in custom DTDs

C.4.1. XHTML Common Names

- 80 -

Modularization of XHTMLC.4.1. XHTML Common Names

C.4.2. XHTML Common Attribute Definitions

- 81 -

C.4.2. XHTML Common Attribute DefinitionsModularization of XHTML "lang %LanguageCode; #IMPLIED xml:lang %LanguageCode; #IMPLIED dir (ltr|rtl) #IMPLIED" >

]]>

C.4.3. XHTML Transitional Attribute Definitions

]]>

- 83 -

C.4.3. XHTML Transitional Attribute DefinitionsModularization of XHTML C.4.4. XHTML Strict Content **Model**

- 84 -

Modularization of XHTMLC.4.4. XHTML Strict Content **Model**

- 85 -

C.4.4. XHTML Strict Content ModelModularization of XHTML

- 86 -

Modularization of XHTMLC.4.4. XHTML Strict Content **Model** C.4.5. XHTML Transitional Content **Model**

- 87 -

C.4.5. XHTML Transitional Content ModelModularization of XHTML ]]>

- 88 -

Modularization of XHTMLC.4.5. XHTML Transitional Content **Model**

]]>

- 89 -

C.4.5. XHTML Transitional Content ModelModularization of XHTML

C.4.6. Intrinsic Events

- 90 -

Modularization of XHTMLC.4.6. Intrinsic Events

- 91 -

C.4.6. Intrinsic EventsModularization of XHTML onblur %Script; #IMPLIED >

]]>

C.4.7. XHTML Character Entities

%XHTML1-lat1;

%XHTML1-symbol;

- 92 -

Modularization of XHTMLC.4.7. XHTML Character Entities %XHTML1-special; ]]>

C.5. XHTML Document Structure Modules This section contains the formal definition of each XHTML Module.

C.5.1. Structure

- 93 -

C.5. XHTML Document Structure ModulesModularization of XHTML ]]>

]]>

- 94 -

Modularization of XHTMLC.5.1. Structure C.5.2. Frameset

- 95 -

C.5.2. FramesetModularization of XHTML

C.5.3. Lists

- 96 -

Modularization of XHTMLC.5.3. Lists

- 97 -

C.5.3. ListsModularization of XHTML -->

]]>

C.6. XHTML Block Element Modules This section contains the formal definition of the Block Element Modules.

C.6.1. Block Structural

- 98 -

Modularization of XHTMLC.6. XHTML Block Element Modules

C.6.2. Block Phrasal

]]>

- 100 -

Modularization of XHTMLC.6.2. Block Phrasal C.6.3. Block Presentational

]]>

- 101 -

C.6.3. Block PresentationalModularization of XHTML C.7. XHTML Inline

Element Modules This section contains the formal definition of each XHTML Inline Element Module.

C.7.1. Inline Structural

- 102 -

Modularization of XHTMLC.7. XHTML Inline Element Modules

C.7.2. Inline Phrasal

- 103 -

C.7.2. Inline PhrasalModularization of XHTML

C.7.3. Inline Presentational

- 105 -

C.7.3. Inline PresentationalModularization of XHTML >

]]>

C.8. XHTML Special Case Modules This section contains the formal definition of each of the XHTML Special Case Modules.

C.8.1. Meta

- 106 -

Modularization of XHTMLC.8. XHTML Special Case Modules

C.8.2. Linking

- 107 -

C.8.2. LinkingModularization of XHTML rel %LinkTypes; #IMPLIED rev % LinkTypes; #IMPLIED accesskey %Character; #IMPLIED tabindex %Number; #IMPLIED >

C.8.3. Image

]]>

C.8.4. Client-side Image Map

C.8.5. Object

]]>

C.8.6. Applet

]]>

- 112 -

Modularization of XHTMLC.8.6. Applet C.8.7. Scripting

C.8.8. Stylesheets

C.8.9. HTML 3.2 Tables

- 114 -

Modularization of XHTMLC.8.9. HTML 3.2 Tables "align (left|center|right) #IMPLIED" >

- 115 -

C.8.9. HTML 3.2 TablesModularization of XHTML

]]>

C.8.10. HTML 4.0 Tables

- 116 -

Modularization of XHTMLC.8.10. HTML 4.0 Tables

- 117 -

C.8.10. HTML 4.0 TablesModularization of XHTML frame %TFrame; #IMPLIED rules %TRules; #IMPLIED cellspacing %Length; #IMPLIED cellpadding %Length; #IMPLIED datapagesize CDATA #IMPLIED >

- 118 -

Modularization of XHTMLC.8.10. HTML 4.0 Tables

- 119 -

C.8.10. HTML 4.0 TablesModularization of XHTML -->

]]>

C.8.11. HTML 3.2 Forms

]]>

]]>

- 121 -

C.8.11. HTML 3.2 FormsModularization of XHTML >

C.8.12. HTML 4.0 Forms

]]> - 122 -

Modularization of XHTMLC.8.12. HTML 4.0 Forms "( %Heading.class; | %List.class; | %Block-noform.mix; | fieldset )+" >

- 123 -

C.8.12. HTML 4.0 FormsModularization of XHTML

]]>

- 124 -

Modularization of XHTMLC.8.12. HTML 4.0 Forms accesskey %Character; #IMPLIED %LegendAlign.attrib; >

C.9. Reformulated XHTML 1.0 DTDs This section defines the three XHTML 1.0 DTDs based upon the modules defined above. Thecontent models of these DTD do not differ from their predecessors in XHTML 1.0 [XHTML1] [p.58] .

C.9.1. XHTML 1.0 Strict

- 126 -

Modularization of XHTMLC.9.1. XHTML 1.0 Strict "XHTML1-arch.mod" > %XHTML1-arch.mod; ]]>

%XHTML1-names.mod; ]]>

%XHTML1-charent.mod; ]]>

%XHTML1-events.mod; ]]>

%XHTML1-attribs.mod; ]]>

%XHTML1-**model**.mod; ]]>

%XHTML1-inlstruct.mod;

- 127 -

C.9.1. XHTML 1.0 StrictModularization of XHTML ]]> %XHTML1-inlpres.mod; ]]>

%XHTML1-inlphras.mod; ]]>

%XHTML1-blkstruct.mod; ]]>

%XHTML1-blkpres.mod; ]]>

%XHTML1-blkphras.mod; ]]>

%XHTML1-script.mod; ]]>

- 128 -

Modularization of XHTMLC.9.1. XHTML 1.0 Strict %XHTML1-style.mod; ]]>

%XHTML1-image.mod; ]]>

%XHTML1-frames.mod; ]]>

%XHTML1-linking.mod; ]]>

%XHTML1-csismap.mod; ]]>

%XHTML1-object.mod; ]]>

- 129 -

C.9.1. XHTML 1.0 StrictModularization of XHTML PUBLIC "-//W3C//ELEMENTS XHTML 1.0 Lists//EN" "XHTML1-list.mod" > %XHTML1-list.mod; ]]>

%XHTML1-form.mod; ]]>

%XHTML1-table.mod; ]]>

%XHTML1-meta.mod; ]]>

%XHTML1-struct.mod; ]]>

C.9.2. XHTML 1.0 Transitional

- 131 -

C.9.2. XHTML 1.0 TransitionalModularization of XHTML

%XHTML1-arch.mod; ]]>

%XHTML1-names.mod; ]]>

%XHTML1-charent.mod; ]]>

%XHTML1-events.mod; ]]>

%XHTML1-attribs-t.mod; ]]>

%XHTML1-**model**-t.mod;

- 132 -

Modularization of XHTMLC.9.2. XHTML 1.0 Transitional ]]> %XHTML1-inlstruct.mod; ]]>

%XHTML1-inlpres.mod; ]]>

%XHTML1-inlphras.mod; ]]>

%XHTML1-blkstruct.mod; ]]>

%XHTML1-blkpres.mod; ]]>

%XHTML1-blkphras.mod; ]]>

- 133 -

C.9.2. XHTML 1.0 TransitionalModularization of XHTML %XHTML1-script.mod; ]]>

%XHTML1-style.mod; ]]>

%XHTML1-image.mod; ]]>

%XHTML1-frames.mod; ]]>

%XHTML1-linking.mod; ]]>

%XHTML1-csismap.mod; ]]>

- 134 -

Modularization of XHTMLC.9.2. XHTML 1.0 Transitional "XHTML1-object.mod" > %XHTML1-object.mod; ]]>

%XHTML1-applet.mod; ]]>

%XHTML1-list.mod; ]]>

%XHTML1-form.mod; ]]>

%XHTML1-table.mod; ]]>

%XHTML1-meta.mod; ]]>

%XHTML1-struct.mod;

- 135 -

C.9.2. XHTML 1.0 TransitionalModularization of XHTML ]]>

C.9.3. XHTML 1.0 Frameset

%XHTML1-t.dtd;

- 137 -

```
SHOW FILES

File  Name
----  -------------
   9: Business & Industry(R)_Jul/1994-2004/Mar 10
  15: ABI/Inform(R)_1971-2004/Mar 10
  16: Gale Group PROMT(R)_1990-2004/Mar 11
 160: Gale Group PROMT(R)_1972-1989
  47: Gale Group Magazine DB(TM)_1959-2004/Mar 11
  88: Gale Group Business A.R.T.S._1976-2004/Mar 10
  98: General Sci Abs/Full-Text_1984-2004/Feb
 141: Readers Guide_1983-2004/Feb
 148: Gale Group Trade & Industry DB_1976-2004/Mar 05
 275: Gale Group Computer DB(TM)_1983-2004/Mar 11
 347: JAPIO_Nov 1976-2003/Nov(Updated 040308)
 369: New Scientist_1994-2004/Feb W5
 484: Periodical Abs Plustext_1986-2004/Mar W1
 553: Wilson Bus. Abs. FullText_1982-2004/Feb
 570: Gale Group MARS(R)_1984-2004/Mar 11
 583: Gale Group Globalbase(TM)_1986-2002/Dec 13
 608: KR/T Bus.News._1992-2004/Mar 11
 613: PR Newswire_1999-2004/Mar 10
 621: Gale Group New Prod.Annou.(R)_1985-2004/Mar 11
 624: McGraw-Hill Publications_1985-2004/Mar 11
 634: San Jose Mercury_ Jun 1985-2004/Mar 10
 635: Business Dateline(R)_1985-2004/Mar 10
 636: Gale Group Newsletter DB(TM)_1987-2004/Mar 11
 647: CMP  Computer Fulltext_1988-2004/Feb W5
 674: Computer News Fulltext_1989-2004/Feb W5
 696: DIALOG Telecom. Newsletters_1995-2004/Mar 10
 810: Business Wire_1986-1999/Feb 28
 813: PR Newswire_1987-1999/Apr 30
  13: BAMP_2004/Feb W5
  20: Dialog Global Reporter_1997-2004/Mar 11
  75: TGG Management Contents(R)_86-2004/Feb W5
 211: Gale Group Newsearch(TM)_2004/Mar 11
 370: Science_1996-1999/Jul W3
 486:  Press-Telegram_1992- 2004/Mar 10
 610: Business Wire_1999-2004/Mar 10
 623: Business Week_1985-2004/Mar 11
 637: Journal of Commerce_1986-2004/Mar 11

?
```

Command | submit | copy/paste selection

Previous commands | s (session(w)state)(s)(xml(w)input .... | show current buffer | show entire buffer

```
S  (SESSION(W)STATE) AND (XML(W)INPUT)

Your SELECT statement is:
   S  (SESSION(W)STATE) AND (XML(W)INPUT)

          Items   File
          -----   ----

   No files have one or more items; file list includes 37 files.

?
```

Command | submit | copy/paste selection

Previous commands: s (session(w)state)(s)(xml(w)input .... | show current buffer | show entire buffer